WS 1991/92

Bericht im ersten praktischen Studiensemester

Hauptthema:

Erstellen eines Data Dictionarys aus
den Assembler-Source-Codes der MS-
Macros des IMOS-PPS-Paketes

Name, Vorname: Feyrer, Hubert

Studiengruppe: I3T

Ausbildungsbetrieb:

UBG
Unternehmensberatung Gommel GmbH
Weißenburgstraße 3
8400 Regensburg

Abgabetermin: 10. Januar 1992

Kenntnis genommen:

.....

(Hr. S. Ligl, Praktikumsbetreuer)

Bestätigung

Hiermit bestätigen wir, daß Hr. Hubert Feyrer, geb. 11. März 1972, wohnhaft in 8304 Mallersdorf, Bachstr. 40, seit dem 9. September 1991 bis zum 10. Janual 1992 sein erstes praktisches Studiensemester in der Software-Abteilung unseres Hauses ableistet.

.....	.....
(Hr. S. Ligl, Praktikumsbetreuer)	(Hr. W. Gommel, Geschäftsführung)

Inhaltsverzeichnis

1 DV-Umgebung der UBG

Bevor näher auf die eigentliche Tätigkeit während des Praktikums — auf die Erstellung des Data Dictionarys — eingegangen wird, soll hier zuerst geschildert werden, in welcher DV-technischen Umgebung sich das Praktikum abspielte.

1.1 Der CTM-Rechner

Der wichtigste Rechner der UBG ist die CTM-Anlage, ein Rechner der Fa. Computertechnik Müller GmbH (heute Itos), Konstanz. Auf ihm werden sämtliche Programme entwickelt. Es handelt sich dabei um einen für wirtschaftliche Anwendungen ausgelegten Prozeßrechner, der unter dem Betriebssystem ITOS (*I*ntelligent *T*erminal *O*perating *S*ystem) läuft. Der Prozessor besitzt einen 16 Bit breiten Adressbus und 4MB Arbeitsspeicher. Für den Mehrbenutzerbetrieb können mehrere intelligente Terminals (Bildschirmarbeitsplätze, BAP) angeschlossen werden.

Auf diesen Terminals mit je bis zu 2,5MB eigenem Arbeitsspeicher ist dann ein Arbeiten mit mehrern (maximal 10) Partitions möglich, d. h. man kann mehrere Programme auf verschiedenen (virtuellen) Bildschirmen gleichzeitig ablaufen lassen und beliebig zwischen den einzelnen Programmen hin- und herschalten. Bei Plattenzugriffen wird über DNÜ (Datennah-Übertragung, lokales Netzwerk) auf den Massenspeicher der Zentraleinheit zurückgegriffen. Dieser besteht aus zwei Festplatten mit je 300MB formatierter Kapazität sowie einer mittleren Zugriffszeit von 30ms.

Das Dateisystem weist keine der herkömmlichen Verzeichnishierarchien wie etwa UNIX oder MS-DOS auf, sondern besteht aus sog. “Bibliotheken”. Dabei wird grundsätzlich zwischen zwei Arten unterschieden: System- und Anwenderbibliotheken. Systembibliotheken existieren etwa in Form der “Swap”-Bibliothek, die für das Speichern ausgelagerter Speicherbereiche verantwortlich ist, oder der Betriebssystem-Bibliothek, die das Betriebssystem in ausführbarer Form enthält.

Eine Anwender-Bibliothek besteht aus drei physikalischen Dateien: Erstens der Source-Datei. Diese enthält die Quellcodes der Programme, die in der zweiten Datei in ausführbarer Form, als Objekt-Datei, stehen. Die dritte Datei schließlich enthält das Inhaltsverzeichnis, in dem Verweise auf die Anfänge aller Dateien in den ersten beiden Dateien stehen. Die MS-Macros, die Informationen für das zu erstellende Programm enthalten, stehen z. B. in Bibliothek 1.

Programmiert wird die CTM-Anlage der UBG in Assembler. Zwar existieren seit einiger Zeit auch Sprachen wie Cobol oder BASIC, bei der Einführung der CTM-Anlage im Jahre 1978 war jedoch Assembler die einzige verfügbare Sprache, so daß auch heute noch jedes Programm in Assembler entwickelt wird, da eine Umstellung zu aufwendig wäre: das IMOS-PPS-Paket besteht z. Zt. aus ca. 400–500 Programmen mit durchschnittlich 1000 Zeilen Code.

1.2 Von CTM nach UNIX

IMOS läuft jedoch nicht nur auf dem CTM-Rechner, sondern auch auf verschiedenen UNIX-Rechnern. Im folgenden wird der Weg eines Programms von CTM nach UNIX beschrieben.

1.2.1 CTM-Polyline

Grundsätzlich werden alle Programme auf dem CTM-System entwickelt und anschießend, wenn sie unter CTM ausgetestet wurden, durch das Programm "SCOERF" für die Transformation nach C/UNIX erfaßt. SCOERF schreibt die Programminformationen zum jeweiligen Programm in die CTM-Datei `IMOS-SPSFREMD`, das Übertragen des Programms wird vom Zielrecher initialisiert.

Transformiert werden können Programe, Bilder, Druckmasken und "normale" Dateien, z. B. mit IMOS angelegte Daten. MS-Macros können nicht einfach erfaßt werden, für sie muß auf UNIX-Seite erst ein funktionales Äquivalent erstellt werden. Dies geschieht in Form eines Include-Files oder einer C-Funktion.

1.2.2 Gateway-/DOS-Rechner

Die nächste Station auf dem Weg nach UNIX ist der Gateway-Rechner. Dies ist ein MS-DOS-Rechner mit Intel i386-Prozessor, der sowohl mit der CTM-Polyline als auch mit dem SCO[1]-UNIX-Rechner hardwaremäßig verbunden ist. Auf der Software-Seite verfügt dieser PC über das *PC-Interface*, ein Produkt der Forma LOCUS, mit dem es möglich ist, auf die Platte eines SCO-UNIX-Rechers zu schreiben. Dieser Recher ist der Übertragung von CTM auf UNIX gewidmet, sonst ist er ohne Funktion. Er kopiert das auf dem CTM-Rechner bereitliegenden File `IMOS-SPSFREMD` auf die Harddisk des SCO-Rechners.

1.2.3 SCO-UNIX

Nachdem die zu übertragende Datei auf die Platte des SCO-Rechners — dieser hat im lokalen (UNIX-)Netzwerk den Namen "SCODEV" — geschrieben wurde, wird sie dort von einem dauernd laufenden Prozess übernommen. Anschließend wird das übertragene Programm "transformiert", d. h. in C-Sourcen umgewandelt und übersetzt[2].

[1] *S*anta *C*ruz *O*peration, benannt nach dem Hersteller dieser zu AT&T-SVR4-kompatiblen UNIX-Version

1.3 Die UNIX-Plattformen der UBG

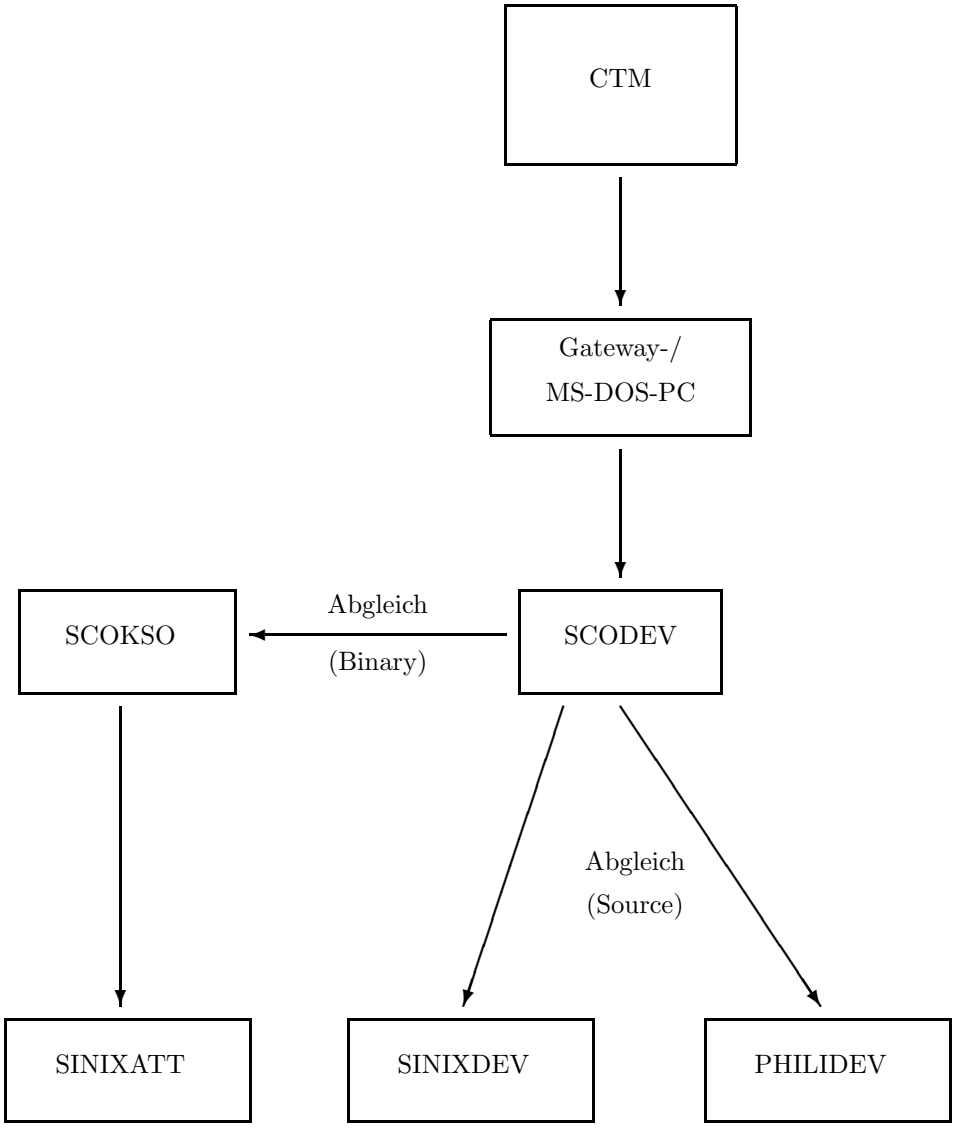
Im Folgenden werden die unterstützten Hardware-Plattformen erläutert, die bei der UBG installiert sind und auf denen IMOS angeboten wird. Der zuerst angegebene Rechnername entspricht dabei dem Namen des jeweiligen Rechners im Netz, die verwendeten Betriebssysteme sind alle konform zum UNIX System V von AT&T.

- SCODEV: Entwicklungsrechner der Marke "Bayern PC" unter SCO-UNIX; Prozessor: Intel i386/33; (s. o.)
- SINIXDEV: Entwicklungsrechner Marke "Siemens-MX300" (NSC-Prozessor), läuft unter SINIX-Betriebssystem bis Version 5.4, d. h. er unterstützt 3 Universen: UCB (BSD[2]-kompatibel), SINIX und ATT; Dieser Rechner wird v. a. für Vorführungen und Schulungen verwendet
- PHILIDEV: Entwicklungsrechner Philips P9050 mit Motorola-Prozessor MC68030
- SCOKSO: KSO[3]-Rechner Bayern-PC mit SCO-UNIX, Prozessor: Intel i386; Dieser Rechner enthält einen gesicherte Stand der IMOS-Programme auf SCO-UNIX. Auf SCODEV können die Programme noch fehlerhaft sein, da sie sich im Entwicklungsstadium befinden.
- SINIXATT: Siemens-Rechner MX300 (Intel i386-Prozessor) mit Betriebssystem SINIX (ab V5.4), nur ATT-Universum

Aufgrund der Binärkompatibilität zwischen den Systemen SCODEV, SCOKSO und SINIXATT brauchen die Programme auf SCOKSO und SINIXATT nicht einzeln übersetzt werde, sondern hier werden die unter SCODEV übersetzten und ausgetesteten Programme in binärer Form übernommen. Dies ist durch das ABI[4] der zu AT&T-SVR4 kompatiblen UNIX-Versionen möglich.

Die folgende Abbildung faßt nochmals den kompletten Weg vom CTM-Rechner zum UNIX-Zielrechner zusammen[3]:

[2] *B*erkeley *S*ystem *D*istribution; neben AT&T-System V bedeutendste UNIX-Strömung
[3] *K*unden–*S*ervice–*O*rganisation
[4] Application Binary Inferface



Außer auf den oben aufgeführten Rechnern wurde IMOS bereits erfolgreich auf mehreren Rechnern mit dem Motorola-RISC-Prozessor MC88000 installiert. Die neueste Anschaffung der UBG besteht aus einem IBM-RISC[5]-Rechner der Serie RS/6000, Typ 320, unter dem Betriebssystem AIX[6] Version 3. Einige Leistungskenndaten[1]:

[5] *R*educed *I*nstruction *S*et *C*omputer; Spezielle IBM-Definition für den vorliegenden Prozessor: *R*educed *I*nstruction *S*et *C*ycles

[6] *A*dvanced *I*nteractive *E*xecutive

- POWER[7]-CPU mit 20MHz
- 8K Instruktionscache
- 32K Datencache
- 27,5 MIPS[8] (basierend auf Dhrystones 1.1, normalisiert auf VAX 11/780 MFLOPS basierend auf Double Precision); Zum Vergleich: Sun SPARC: 17 MIPS, Motorola MC68040/25: 19,7 MIPS, Intel i486SX/25: 15 MIPS, Intel i486DX/50: 41 MIPS,
- 7,4 MFLOPS[9]

2 Anforderungen an den Macro Scanner

Das von der Firma UBG produzierte Produktions-Planungs- und -Steuerungs-System (PPS-System) "IMOS" enthält eine Vielzahl verschiedener Datenstrukturen, sog. "Satzarten", in denen die Strukturen der zu verwaltenden Daten gespeichert werden. Die Satzarten werden in Include-Dateien, den "MS-Macros", definiert. Im Prinzip kann die Form, in der eine Satzart definiert wird, mit einer `struct`-Anweisung der Sprache C oder einem `record` in PASCAL verglichen werden, jedes Satzart-Feld entspricht dabei einem Feld der `struct`/des `record`s.

Die zu erfüllende Aufgabe besteht nun darin, ein Programm zu schreiben, das aus den MS-Macros ein *Data Dictionary* erstellt, d. h. es soll ein "Daten-Wörterbuch" mit Informationen über die vorhandenen Datenstrukturen erstellt werden. Die im Data Dictionary gespeicherten sog. *Meta-Daten* enthalten also "Daten über Daten"[4]. Das Data Dictionary sollte zum einen allgemeine Informationen wie etwa Satzlänge, Zugriffsart oder Schlüssellänge über jede vorhandene Satzart enthalten. Zum anderen werden Daten über die einzelnen Satzart-Felder erfaßt: Feldbreite (in Bytes), (Daten)Typ, ...

Das erzeugte Data Dictionary ist Teil des UBG-CASE[10]-Konzeptes. Es soll das Verzeichnis der Satzart-Verwendung ergänzen, in dem erfaßt wurde, in welchem Programm auf welche Satzart zugegriffen wird.

Weitere Bausteine des CASE-Konzeptes sind:

- Die oben beschriebene automatische Transformation der Assembler- in C-Quellcodes

[7] Performance Optimized With Enchanted RISC
[8] *M*illion *I*nstructions *P*er *S*econd
[9] *M*illion *FL*oating-point-*O*perations *P*er *S*econd
[10] Für den Beriff "CASE" existieren mehrere Definitionen: *C*omputer *A*dded *S*oftware-*E*ngineering, *C*omputer-*A*dded *S*ystem-*E*ngineering, *C*omputer-integrated *A*pplication-*S*ystem-*E*ngineering; Die letztere trifft hier am besten zu

- IMOS-Dokumentations-System. Der weit verbreitete Stand der Software-Dokumentation sieht so aus, daß die Dokumentation nicht dem Softwarestand entspricht, außerdem unvollständig ist und meist zu spät ausgeliefert wird.

  Die Lösung der UBG lautet hier: 100%ige Vorausdokumentation. Aus der Tatsache, daß jedes Wissen sofort dokumentiert wird, ergeben sich für den Kunden zwei entscheidende Vorteile:

  Zum einen wird die Dokumentation vor bzw. mit der Software ausgeliefert. Dazu kommt, das die Dokumentation stets auf dem aktuellsten Stand ist, was dem Kunden die Arbeit mit IMOS zusätzlich erleichtert, da bei evtl. auftretende Probleme sofort die richtigen Unterlagen bereitliegen.

Langfristig ist die Erstellung einer Entwicklungsdatenbank bzw. eines Reprositorys mit Datenbank-Anschluß geplant, das bei der Programmentwicklung hinzugezogen werden kann wie etwa das Satzart-Handbuch, das z. Zt. noch verwendet wird. Das Satzart-Handbuch enthält ein Verzeichis aller existierenden Satzarten, es soll in absehbarer Zeit durch den Output des Programms “DSS04” ersetzt werden.

Eine weitere Verwendung des Data Dictionary ergibt sich bei der Ausgabesteuerung: hier bräuchte nur noch angegeben werden, welche Felder angezeigt werden sollen, die weiteren Informationen wie die Anzahl der Vor- und/oder Nachkommanstellen würden dann aus dem Data Dictionary geholt werden. Daraus resultiert, das z. B. Bildschirmmasken beim Kunden abgeändert werden können und sofort — ohne das jeweilige Programm wieder neu zu übersetzen — einsatzbereit sind.

Auch wird die Verwendung des Data Dictionary eine verstärkte Zentralisierung und somit eine geringere Redundanz in der Datenhaltung mit sich bringen, da die Informationen nur noch einmal, im Data Dictionary, vorkommen. In den jeweiligen Programmen bräuchten dann nur noch Verweise auf die jeweilige Satzart und das entsprechende Feld zu stehen, ohne die Daten ein zweites mal zu halten.

Die Wartung des Data Dictionary besteht darin, daß — falls ein MS-Macro geändert wird — die geänderten Daten sofort auch in das Data Dictionary aufgenommen werden, damit alle Programme, die sich auf das Data Dictionary stützen, up to date sind.

Neben dem eigentlichen Programm zum Erfassen der MS-Macros sollten außerdem drei S-Programme (Satzarten-Programme) programmiert werden, die dazu verwendet werden können, um die Sätze des Data Dictionary auch betrachten und (manuell) ändern zu können sowie zur Kontrolle, ob der eigentliche Macro-Scanner korrekt arbeitet.

3 Konzeption des Macro-Scanners

Das Data Dictionary beruht auf den drei folgenden Datenstrukturen:

1. Satzart S9006 (entsprechendes Macro: M9006): Informationen zu den Satzart-Köpfen:

S9006K1 . . . . Satzart
S9006K2 . . . . Versionskennzeichen; ‘␣’=aktuell, ‘A’=alt
S9006BE . . . . Satzartenbezeichnung
S9006DT . . . Datum der letzten Änderung; Format: TTMMJJ
S9006DA . . . Dateiname
S9006KZ . . . . Kennzeichen ‘I’=intern, ‘␣’=extern
S9006TY . . . Dateityp; ‘I’=ISAM, ‘S’=SAM
S9006SL . . . . Satzlänge
S9006KL . . . . Schlüssellänge
S9006KP . . . Schlüsselposition
S9006M1 . . . Satzart Matchcode 1
S9006M2 . . . Satzart Matchcode 2
S9006M3 . . . Satzart Matchcode 3
S9006M4 . . . Satzart Matchcode 4
S9006M5 . . . Satzart Matchcode 5
S9006M6 . . . Satzart Matchcode 6
S9006MN . . . Macro-Name

2. S9007 im Macro M9007: Informationen über die einzelnen Satzart-Felder:

   S9007K1 . . . . Satzart
   S9007K2 . . . . Versionskennzeichen; ‘␣’=aktuell, ‘A’=alt
   S9007K3 . . . . Feldnummer
   S9007SY . . . Symbolbezeichnung
   S9007F1 . . . . Feldbezeichnung 1
   S9007F2 . . . . Feldbezeichnung 2
   S9007ST . . . . Stellen
   S9007TY . . . Feldtyp
   S9007BY . . . Bytes
   S9007PO . . . Position im Satz
   S9007KZ . . . Kennzeichen Erläuterungstext (J/N)
   S9007AN . . . Anzahl Nachkommastellen
   S9007GK . . . Gruppierungskennzeichen

3. S9008 im Macro M9008: Erläuterungn zu den Satzart-Feldern:

   S9008K1 . . . Satzart

S9008K2 ... Versionskennzeichen; '␣'=aktuell, '`A`'=alt

S9008K3 ... Feldnummer

S9008K4 ... Laufende Nummer

S9008ER ... Erläuterung

Diese Satzarten waren mit den entsprechenden Daten zu belegen, die einzelnen Sätze wurden in den Dateien "`IMOS0000DSSSAKOP`" (S9006, Datensatz-Strukturen/Satzart-Kopf), "`IMOS0000DSSSAFEL`" (S9007, Satzart-Feld) und "`IMOS0000DSSSAERL`" (S9007, Satzart-Erläuterungsn) abgelegt.

Noch ein Wort zum Aufbau der Macro-Namen und Satzart-Feldbezeichnungen. Jedes Macro (jede Satzart) erhält eine (normalerweise) eindeutige dreistellige Nummer, die bei Macros dem Präfix "`MS`" folgt. Bei Satzarten ist dieser Präfix entsprechend "`SA`". Sollte die jeweilige Nummer 4 Stellen haben, dies ist z. B. bei Work-Dateien der Fall, so wird jeweils der zweite Buchstabe des Präfixes weggelassen. Daraus ergeben sich bei den Satzarten des Data Dictionary die Macro-Namen "`M9006`", "`M9007`" und "`M9008`" bzw. die Satzart-Bezeichnungen `S9006`, `S9007` und `S9008`.

Um die oben genannten Informationen zu gewinnen, wird jedes MS-Macro in mehrere funktionale Blöcke unterteilt, aus denen dann die entsprechenden Daten gewonnen werden.

Diese Blöcke ergeben sich wie folgt:

1. Kopf der Macros-Datei: Zeilen 1 und 2
2. Name des Macros: Zeile 3
3. Satzart-Bezeichnung: 2. Kommentarzeile
4. Die folgenden Blöcke dürfen in (fast) beliebiger Reihenfolge im Macro erscheinen:
   (a) File-Label: Wortsymbol "`DFFLAB`"
   (b) Satzart-Definition: `<label>  DS  ...` bis `ORG`
   (c) Unterprogramme, z. B. Löschroutine für Satzart-Felder (in jedem Macro enthalten!): Von "`SUB`" bis "`RET`"
   (d) Drucktabelle (optional): "`<label>XXX`" bis `ANOP`
   (e) Matchcodes (S9006M1–S9006M6): "`AIF...MC`x `NE 'Y'...`"; $x \in [1;6]$

Das Programm "`DSS04`" ist modular aufgebaut, entsprechend existiert für jeden der aufgeführten Blöcke ein eigenes Unterprogramm, das die jeweiligen Informationen aus ihm herausliest.

Wie bereits weiter oben erwähnt, soll das Programm komplett in Assembler geschrieben werden. Da sich der Prozessor des CTM-Rechners in einigen Punkten von den gängigen

Motorola- und Intel-Prozessoren unterscheidet, seien deshalb hier einige Besonderheiten und die Konsequenzen, die sich daraus ergeben, aufgeführt:

1. Datentypen: Es existieren nur zwei unterschiedliche Datentypen. Dies sind zum einen C-Felder. Dabei handelt es sich um Speicherbereiche (Größenordnung: 1 bis max. 136 Bytes), die in einem abgewandelten ASCII-7-Bit-Zeichensatz, dem IOS-7-Bit-Code, codiert sind. Sie werden Normalerweise zur Speicherung und Verarbeitung von Zeichenketten (Strings) verwendet. Im Gegensatz zu manch anderen String-Formaten besitzen sie weder eine Längenangabe im ersten Byte (vgl. PASCAL) bzw. ein besonderes Zeichen am Stringende (in C '
0').

   Der zweite verwendete Datentyp sind die P-Felder. Dabei handelt es sich um eine gepackte (packed→P-Feld) Darstellungsform, die zur Speicherung von Zahlen verwendet wird. Diese P-Felder besitzen jedoch keine Gemeinsamkeit mit der BCD[11]-Darstellung!

   Die Felder enthalten im ersten Byte, dem Längenbyte, Informationen über die Stellenzahl, das LSB[12] enthält das Vorzeichen: ein '0' bedeutet hier "positiv", ein gesetztes Bit ('1') entspricht einem negativen Vorzeichen. Die Bits 1–7 enthalten die Anzahl der auf das Längenbyte folgenden Bytes, die zum jeweiligen P-Feld gehören.

   Beispiel: Das Längenbyte eines insgesamt 5 Bytes langen, negativen P-Feldes ergibt sich wie folgt:

`1`	Negatives Vorzeichen
`0000100`	4 (5 Bytes − Längenbyte) in Binärdarstellung
`00001001`	Binärcode des vollständigen Längenbytes

   Die folgenden (im obrigen Beispiel 4) Bytes enthalten dann die Zahl in gepackter Darstellung. Die Stellenzahl entspricht dabei der Byte-Länge mal 2, da in einem Byte zwei Dezimalzahlen gespeichert werden. Die Ziffern werden jedoch nicht — wie beim BCD-Code — in jeweils einer Tetrade (4 Bit) angelegt, sondern jeweils zwei Dezimalziffern werden als Zahl zwischen 0 und 99 interpretiert und entsprechend binär codiert abgelegt.

   Beispiel: Die Zahl 1234567 würde wie folgt codiert:

	`1`	`23`	`45`	`67`
BCD (hex):	`01`	`23`	`45`	`67`
P-Feld (hex):	`01`	`17`	`2d`	`43`

   Die Zahl −123456 würde also als 8-stelliges P-Feld wie folgt aussehen (Bytes hexadezimal codiert): 09 01 17 2d 43.

   Ein weiterer Typ, der jedoch nur selten benutzt wird, ist das H-Feld. Es ähnelt dem P-Feld, nur das ein H-Feld ohne dem Längenbyte auskommt.

[11] *B*inary *C*oded *D*ecimal
[12] *L*ease *S*ignificant *B*it, niederwertigstes Bit eines Bytes/Wortes

2. 64 memory-mapped Register: Der CTM-Rechner besitzt 64 16-Bit-Register, die jedoch nicht in der CPU verdrahtet sind, sondern ab Adresse 0 im Arbeitsspeicher. Folglich kann man die Register auch indirekt über ihre Adresse manipulieren: Adr= $2x$, wobei x eine Nummer zwischen 0 und 63 ist, die der Nummer des Registers entspricht.

   Die Register können außerdem dazu verwendet werden, um Felder zu indizieren, was vor allem bei der Stringverarbeitung von Nöten ist.

3. 64K Userspace: Jedem Programm stehen jeweils 64K Speicher zur Verfügung. In diesen 64K müssen sowohl Code als auch Daten untergebracht werden. Ein Heap o. ä. existiert nicht. Werden mehr als 64K benötigt, so müssen Programmteile ausgelagert werden, was jedoch beim vorliegenden Projekt nicht nötig war.

4. Fehlender Stack: Der Prozessor der CTM-Anlage besitzt keine Stackverwaltung. Eine der beiden Konsequenzen, die sich daraus ergeben ist, daß Register nicht auf den Stack zwischengespeichert werden können. Statt dessen müssen sie im Speicher abgelegt werden. Aufgrund der Tatsache, daß die Register memory-mapped sind, ist dies jedoch mit dem kopieren des entsprechende Speicherblockes abgetan.

   Das zweite Problem, das sich aus dem fehlenden Stack ergibt, betrifft den Aufruf von Unterprogrammen. Da hier kein zentraler Mechanismus existiert, um die Rücksprungadressen zu sichern, wurde ein anderer Weg gewählt.

   Da der unter CTM verwendete FAST-Assembler der Fa. HD Software GmbH die erstellung von Macros erlaubt (→MS-Macros!), ist dies relativ transparent, d. h. für den Benutzer der Macros unsichtbar, zu realisieren.

   Die für einen Unterprogramm-Aufruf nötigen Befehle sind, mit den entsprechenden Macros formuliert:

HP		UP
`CALL UP`	⟶	`UP SUB`
		⋮
⋮	⟵	`RET UP`

   Werden die Macros expandiert, so ergibt sich folgender Befehlsablauf:

HP		UP	
`MVC YUP+3,X5,2`			
`BY UP,X5`	⟶	`UP`	`MVC YUP+1,X5,2`
			⋮
			`MVC X5,YUP+3,2`
		`YUP`	`B $$YUP`
⋮	⟵	`<YUP+3>`	`DS L2`

   (Der `MVC`-Befehl hat die selben Parameter wie die C-Funktion "`strcpy()`": Ziel, Quelle, Anzahl zu kopierender Zeichen).

Zuerst wird der Inhalt des Registers X5 (2 Byte) in `YUP+3` gesichert. Dann wird das Unterprogramm aufgerufen, die Rücksprung-Adresse steht zu beginn des Unterprogramms in X5. Diese Adresse wird sodann in den Befehlscode des "`B`"-(Branch)-Befehls beim Label "`YUP`" eingetragen: Der Befehl besteht aus einem Befehlsbyte, gefolgt von 2 Bytes (durch `$$` erzwungen), die die Zieladresse des "`B`"-Befehls angeben. In diese beiden Bytes wird die Rücksprung-Adresse eingetragen.

Diese Anwendung von selbstmodifizierendem Code erklärt auch die vermeintliche Endlosschleife beim Label "`YUP`".

Nachdem das Unterprogramm angearbeitet wurde, wird zuerst die alte Belegung des Registers X5 wieder ergestellt und dann zum aufrufenden Programm zurückgesprungen.

5. Macro-Programmierung: Die oben angesprochene Macro-Techik wird aber nicht nur bei Unterprogramen benutzt, vielmehr ist die Verwendung von Makros Gang und Gäbe. Es existieren Macros zur Bildschirmsteuerung, für die Datei-Ein- und Ausgabe, Druckeransteuerung etc.

   Desweiteren existieren Macros, die zum einen die strukturierte Programmierung unterstützen und zum anderen die transformation nach C erleichtern, da diese Kontrollstrukturen in original-C-Befehle umgewandelt werden können. Die folgenden Macros existieren: `if`/`then`/`else`, `while`-Schleife und `switch`/`case`-Kontrollblock.

6. Langsamer Assembler: Hier halfen nur eiserne Nerven, wenn aufgrund mangelnder Dokumentation nach dem "Trial and Error"-Verfahren programmiert werden mußte.

4 Implementierung des Macro-Scanners

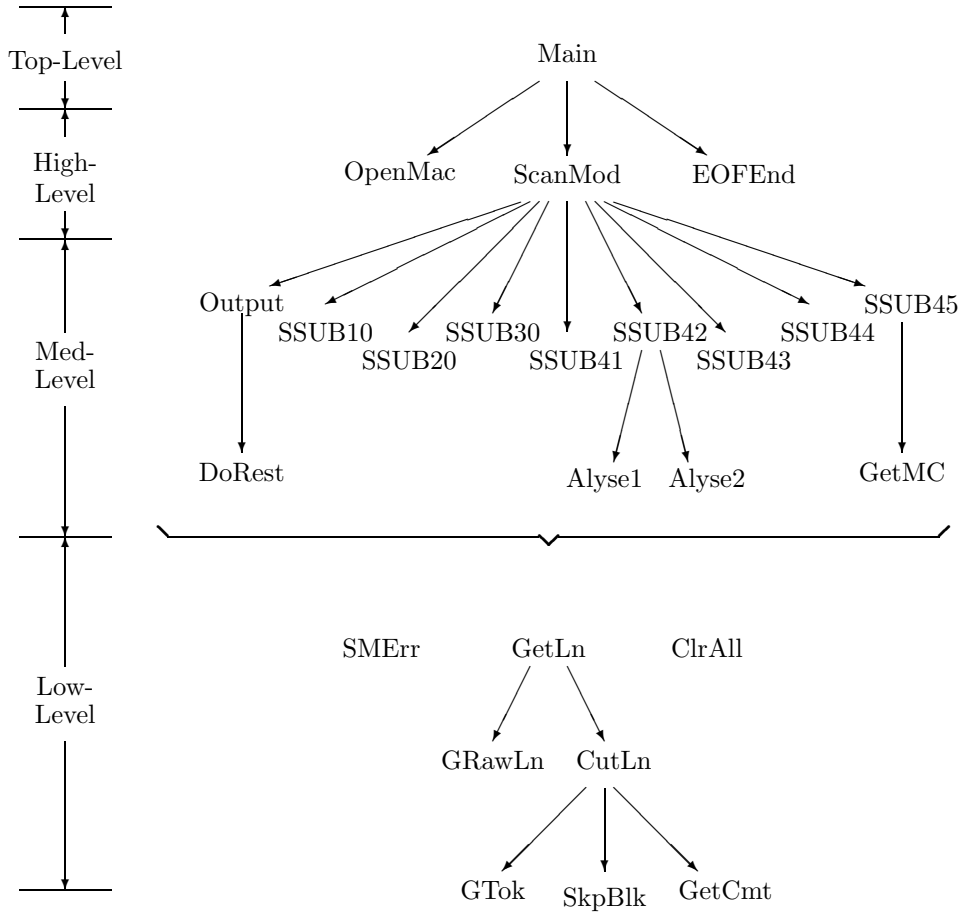
Wie oben erwähnt, existiert für jeden Block, aus dem sich ein Macro zusammensetzen kann, eine eigene Funktion, die den jeweilige Block analysiert. Die Namen dieser Unterprogramme beginnen alle mit dem Präfix "`SSUB`" (*S*can*Sub*).

Neben diesen Funktionen existieren jedoch noch zusätzliche Hilfsfunktionen, die die jeweilige Block-Analyse-Funktion (`SSUB`...) unterstützt bzw. wiederkehrende Abläufe zentralisiert.

Außerdem existieren noch mehrere Low-Level-Funktionen, die für die Datei- Ein/Ausgabe verantwortlich sind, die einzelnen Zeilen aus den Macros lesen, und diese — falls eine Fortsetzungszeile existiert — aneinanderhängt, die Felder und Tabellen, die pro Macro aufgebaut werden, löschen, oder eine zentral angelegte Routine zur Fehlerbehandlung.

Desweiteren existiert eine High-Level-Funktion "`SCANMOD`", die für jedes Macro einzeln aufgerufen wird und die die einzelnen Strukturblöcke zum einen erkennt und das entsprechende `SSUB` aufruft, zum anderen aber auch feststellt, falls eine Struktur mehrmals auftritt und ggf. eine Fehlermeldung ausgibt.

Eine Hierarchie aller Funktionen sähe wie folgt aus:



Es folgt nun die Beschreibung zu allen Funktionen bzw. Unterprogrammen. Im gesamten Programm werden die folgenden Felder und Tabellen aufgebaut und referenziert:

BIBNUM..... Nummer der einzulesenden Macrobibliothek

CCT ..... 128 Zeichen langes C-Feld. Wird der ASCII-Wert eines Zeichens als Index genommen, so kann mit Hilfe dieser Tabelle ermittelt werden, welcher Zeichenklasse das jeweilige Zeichen angehört. Die wichtigsten dieser Zeichenklassen sind:

- B ... Ziffern
- C ... Buchstaben
- N ... Leerzeichen

CFELD ..... Hilfsfeld

CNT ..... Anzahl der belegten Felder in der SA-Feld-Tabellen T7xx

CNT2 ..... Anzahl belegter Erläuterungs-Felder T8xx

D7, D7xx..... Speicher für Bildschirmausgabe der Werte T7xx

D8, D8xx..... Dito für Werte T8xx

DATUM ..... Jüngstes Datum des aktuellen Macros

DATUM2 ..... Änderungsdatum der zuletzt gelesenen Zeile

ERRBUF ..... 135 Zeichen langer Puffer, in dem die Fehlermeldungen zusammengebaut werden

FLAGFT..... Flag, ob File-Table eingelesen

FLAGSA..... Flag, ob Satzarten-Definition definiert

FLAGDR ..... Flag, ob Drucktabelle gelesen

FLAGPR..... Flag, ob Protokolldruck erwünscht, J/N

FLAGMx ..... Flag, ob Matchcode x ($1 \leq x \leq 6$) definiert

LEER..... Mit 150 Blanks gefülltes C-Feld

NAMEBIS .... Name des letzten zu erfassenden Macros

NAMEVON... Name des ersten zu erfassenden Macros

PFELD ..... 4-stelliges Hilfs-P-Feld

PFELD2..... 2-stelliges Hilfsfeld

STACK ..... Zwischenspeicher für Register, CTM hat keinen “echten” Stack!

SYM..... Label der Satzart. Dieses ist im Label jedes Satzart-Feldes vorhanden

SYMLEN ..... Länge des SYM-Labels

T6, T6xx..... Zwischenspeicher für Satzart S9006

T7xx..... Tabellen mit Werten von Satzart S9007

T8xx..... Tabellen mit Werten von Satzart S9008

TOK1..... Token für Labels

TOK2..... Token für Befehle

TOK3..... Token für Befehlsargumente

TOK4..... Token für Kommentare

TOKL1 ..... Länge TOK1

TOKL2 ..... Länge TOK2

TOKL3 ..... Länge TOK3

TOKL4 ..... Länge TOK4

ZEILE ..... Aktuelle logische Zeile aus Macro-Datei

ZEILE2 ..... Aktuelle Zeile aus Macro-Datei

ZNR ..... Nummer der aktuellen Zeile im laufenden Macro

4.1 Top-Level-Funktion

Als erstes wird hier die Nummer der einzulesenden Macro-Bibliothek in BIBNUM gespeichert und die Namen der Macros, von und bis zu denen gelesen werden soll, in den C-Feldern NAMEVON und NAMEBIS abgelegt. Außerdem wird abgefragt, ob ein Protokoll auf den Drucker ansgegeben werden soll. Alle Angaben außer "J" (Ja) werden als "N" (Nein) interprtiert.

Dann werden per Funktionsaufruf an OPENMAC die Ein- und Ausgabedateien geöffnet.

Als nächstes wird die erste Zeile des jeweiligen Macros eingelesen, nachdem deren Position mittels einer indexsequentiellen Datei gefunden wurde. Die eigentliche Hauptaufgabe — die Analyse der Macros — wird dann durch die Funktion SCANMOD erledigt. Ist das Macro zu Ende gelesen und SCANMOD in das Hauptmodul zurückgesprungen, wird ein Verweis auf das nächste Macro der Bibliothek geholt. Ist dessen Name, der in PRLABEL+1 steht, nicht (alphabetisch) größer als NAMEBIS, so wird wiederum an den Anfang der Hauptschleife gesprungen, die erste Zeile des Macros gelesen und dann SCANMOD aufgerufen, etc.

4.2 High-Level-Funktionen

OPENMAC: Dateien und Sätze öffnen

Hier wird ggf. der Drucker (nur bei Protokolldruck) sowie die Ausgabedateien für die Sätze S9006 bis S9008 geöffnet. Zum anderen werden die beiden Eingabedateien geöffnet. Es ist deshalb nötig, zwei Dateien zu öffnen, um an die MS-Macros zu gelangen, da die eigentliche Macro-Datei (-Bibliothek, alle MS-Macros stehen in einer physikalischen Datei!) sämtliche Macros als Text-Blöcke enthält. Die zweite geöffnete, index-sequentielle Datei enthält die Verweise auf die Anfänge der einzelnen Macros in der Gesamtdatei.

Tritt beim Öffnen einer Datei ein Fehler auf, so werden alle bis dahin geöffneten Dateien in EOFEND wieder geschlossen.

Verwendete Register:

- Keine!

EOFEND: Dateien und Sätze schließen

Hier werden die geöffneten Ein- und Ausgabedateien wieder geschlossen. Außerdem wird, bevor der Drucker freigegeben wird, eine Meldezeile ausgegeben, die die Strings NAMEVON und NAMEBIS enthält. Diese Zeile wird nur ausgeegeben, falls Protokolldruck am Anfang auf "J" gesetzt wurde.

Verwendete Register:

- Keine!

SCANMOD: Macro durchlesen

Diese Routine ist das eigentliche Kernstück des ganzen Programmes. Hier werden die einzelnen Blockarten erkannt und entsprechende Funktionen aufgerufen, die den jeweiligen Block verarbeiten.

Zuvor werden hier noch die zu füllenden Datenfelder und Flags in CLRALL initialisiert.

Es gibt die folgende Blöcke, aus denen die Informationen des Data Dictionarys gelesen werden können:

	Block	Funktion
1.	Zeile 1 oder 2	SSUB10
2.	3. Zeile: Macro-Name	SSUB20
3.	2. Kommentarzeile: SA-Bezeichnung	SSUB30
4.	DFFLAB: Filetable	SSUB41
5.	Satzarten-Definition	SSUB42
6.	Routinen (Löschroutine, ...)	SSUB43
7.	Drucktabelle	SSUB44
8.	Matchcode-Definition	SSUB45

Die ersten drei Unterprogramme werden nacheinander aufgerufen, die Informationen müssen also auch in dieser Reihenfolge im Macro stehen. Die nachfolgenden Blöcke (SSUB4x) können in beliebiger Reihenfolge auftreten, es ist jedoch darauf zu achten, daß z. B. die SA-Definition vor der Drucktabelle auftritt.

Im Weiteren liest SCANMOD das Macro zeilenweise ein (nachdem am Anfang bereits einige Zeilen durch SSUB10-30 gelesen wurde), wobei beim Einlesen einer Zeile über GETLN die vier C-Felder TOK1 bis TOK4 mit den einzelnen Spalten der Zeile gefüllt sind (TOK1=Label oder leer; TOK2=Befehl; TOK3= Argument; TOK4=Kommentar oder leer; siehe CUTLN auf Seite 30). Dann wird überprüft, ob die jeweilige Zeile den Anfang eines der oben aufgeführten Blöcke ist bzw. ob irgendwelche Werte gesetzt werden

können, ohne daß weitere Zeilen durch eine spezielle Funktion gelesen werden müssen. Im Einzelnen werden die folgenden Fälle überprüft:

- File-Label: Tritt ein Befehl (TOK2) "`DFFLAB`" auf, so wird SSUB41 aufgerufen und das globale Flag FLAGFT[13] auf 2 gesetzt. Es gibt aber auch Macros, die keine eigene Filetable enthalten, sondern in der ersten `DC`-Anweisung die Adresse einer (externen) Filetable enthalten. Zur Zeit werden die drei File-Labels "`TIMOS`", "`TIMST`" und "`T102`" erkannt. Ihre Werte stehen fest im Programm.

  Ist keine Filetable vorhanden und steht in der ersten "`DC`"-Anweisung des Macros keines der obrigen Schlüsselwörter, so wird ein Fehler Nr. 01.01 ausgegeben.

- Satzarten-Definition: Wird eine Zeile der Form "`<label> DS 0...`" gefunden, so wird davon ausgegangen, daß der Anfang einer Satzarten-Definition gefunden wurde und das Label in SYM, seine Länge TOKL1 in SYMLEN abgelegt. Anschließend wird SSUB42 aufgerufen sowie FLAGSA auf 2 gesetzt. Fehlt die "`0`", wird also nur ein Label und "`DS`" angegeben, so wird davon ausgegangen, daß nur Speicher reserviert wird und keine Elemente deklariert werden, weshalb FLAGSA sofort auf 2 gesetzt wird. Diese Form der Satzarten-Definition kommt oft in Matchcode-Macros vor.

- Unterprogramme: Diese werden am Befehl "`SUB`" erkannt. Die folgenden Zeilen werden bis zum UP-Ende, d. h. bis TOK2 gleich "`RET`" ist, überlesen.

- Drucktabelle: Diese wird durch eine Zeile "`<label>XXX DC ...`" eingeleitet. Ist zu diesem Zeitpunkt FLAGSA=0, d. h. wurde noch keine Satzart definiert, so wird ein Fehler #04.02 ausgegeben. Ansonsten wird SSUB44 aufgerufen und FLAGDR auf 2 gesetzt.

- Matchcodes: Diese sind durch die Assemblerdirektive "`AIF`", gefolgt von einer Abfrage auf "`Y`" gekennzeichnet. Wird eine solche Zeile gefunden, so wird SSUB45 aufgerufen.

Wird in einer Zeile "`MODEND`" gefunden, dann ist der Macro-Text zu Ende und die gelesenen Daten werden in OUTPUT in die jeweiligen Sätze abgestellt.

Diese Routine würde — in vereinfachtem C — etwa wie folgt aussehen:

```
SCANMOD()
{
    CLRALL();               /* alles loeschen */
    SSUB10();               /* Konsistenz-Check */
    SSUB20();               /* Macro-Name */
    SSUB30();               /* Satzarten-Bezeichnung */

SM2:
    GETLN();                /* Naechste Zeile lesen */
```

[13] siehe Punkt 4 auf Seite 14

```
/* File-Label */
if( FLAGFT!=2 && ( TOK2=="DFFLAB" || TOK2=="DC" ) ){
    if( TOK2=="DC" ){
        if( TOK3=="TIMOS" ){
            /* T6xx mit Werten fuer TIMOS belegen */
            FLAGFT=2;
        }else if( TOK3=="T102" ){
            /* T6xx mit Werten fuer T102 belegen */
            FLAGFT=2;
        }else if( TOK3=="TIMST" ){
            /* T6xx mit Werten fuer TIMST belegen */
            FLAGFT=2;
        }else
            SMERR( 01.01 );       /* Unbekannte DFFLAB */
    }else if( TOK2=="DFFLAB" ){
        SSUB41();
        FLAGFT=2;
    }
}

/* SA-Definition */
if( FLAGSA!=2 && TOKL1!=0 && TOK2=="DS"){
    if( TOK3[0]=='0' )
        SSUB42();
    FLAGSA=2;
}

/* Unterprogramme */
if( TOK2=="SUB" ) SSUB43();

/* Drucktabelle */
if( TOK1="...XXX" ){
    if( FLAGSA==2 ){
        SSUB44();
        FLAGDR=2;
    }else{
        /* Drucktabelle vor SA-Definition */
        SMERR( 04.02 );
    }
}

/* Matchcode */
if( TOK2=="AIF" && TOK3[10]=='Y') SSUB45();

/* Ende */
```

```
    if( TOK2=="MODEND" ){
        /* Alle erwarteten Bloecke da? */
        if( FLAGSA==0 ) SMERR( 05.01 );
        if( FLAGFT!=2 ) SMERR( 05.02 );

        OUTPUT();  /* Springt in's MAIN zurueck */
    }

    goto SM2;

}
```

Da die für die Verarbeitung von Zeichenketten nötige Indizierung von Feldern nicht mit den Macros für die strukturierte Programmierung funktioniert, wurde auf jegliche Verwendung dieser Macros verzichtet. Statt dessen wurde auf Verzweigungs- (BY, BN) und Sprunganweisungen (B) zurückgegriffen.

An Registern werden in SCANMOD nur X50 bis X52 für die Fehlermeldungen benutzt.

4.3 Med-Level-Funktionen

OUTPUT: Ausgabe in Dateien

Diese Routine ruft zuerst DOREST auf, um die restlichen Felder, die aus bereits vorhandenen Daten abgeleitet werden müssen, zu füllem. Als nächstes wird das Datum in T6DT, das die Form "JJJJMMTT" hat, was für den Vergleich auf das aktuelle Datum nötig ist, auf die Form "TTMMJJJJ" gebracht.

Dann werden sämtliche Felder T6xx in den Satzartenpuffer S9006 kopiert und überprüft mit dem READ-Macro, ob bereits ein Satz mit dem selben Schlüssel existiert. Ist dies der Fall, so wird der Satz mittels UPD ü"berschrieben, ansonsten wird er mit WRITE neu angelegt, nachdem jeweils T6xx erneut nach S9006 kopiert wurde. Dieses zweite Kopieren ist nötig, da beim lesen mit READ evtl. die in S9006 stehenden Daten überschrieben werden, falls ein Satz gefunden wird.

Als nächstes werden alle Felder, mit Index 0 beginnend, ebenso ausgegeben: zuerst wird T7xx nach S9007 kopiert, dann wird nach der oben beschriebenen Prozedur überprüft, ob der Satz bereits auf Platte existiert und entsprechend geUPDated oder geschrieben (WRITE).

Zuletzt werden die Elemente aus T8xx nach S9008 kopiert und nach der obrigen Methode angespeichert. Es werden soviele Elemente ausgegeben, wie in CNT2 angegeben.

Zusätzlich zur Ausgabe in die jeweilige Datei werden die Sätze T7xx und T8xx nach D7xx und D8xx kopiert. Da die Adresswerde dieser Felder fest sind und nicht in einer Tabelle stehen, können sie über DPFTC/DPFTP parallel zur Ausgabe in Datei auf dem

Bildschirm angezeigt werden.

Mit der oben beschriebenen Vorgehensweise werden so alle Felder ausgegeben, bis der Index-Zähler X50 den Wert CNT (maximaler Index+1, wird in SSUB42 gesetzt und steht hier in X51) bzw. CNT2 erreicht hat. Dann wird noch der Bildschirm gelöscht und das nächste Macro kann eingelesen werden, nachdem zusätzlich eine "`OK`"-Meldung ausgegeben wurde.

Register:

- X50... Indexzähler
- X51... höchster Index+2 (CNT)
- X52... nicht benutzt!
- X53... für Index-Offset-Umrechnungen

DOREST: Restliche Werte berechnen

Hier werden die Werte, die nicht direkt aus den MS-Macros gelesen werden können, aus den bereits vorhandenen Daten errechnet. Im einzelnen sind dies:

- Stellenzahl T7ST:

$$\text{T7ST} = \begin{cases} 2 \cdot \text{T7BY} & \text{“F”} \\ 2 \cdot (\text{T7BY} - 1) & \text{bei T7TY} = \text{“P”} \\ \text{T7BY} & \text{“C”, “H”} \end{cases}$$

- Position im Satz T7PO: Dies ist die Summe aller vorhergehenden Feldbreiten T7BY.
- Außerdem müssen die Felder T8K2 bis T8K4 aus den entsprechenden T7-Werten übernommen werden. T8K1 kann erst in OUTPUT belegt werden.

Zuerst wird das Register, das die Position im Satz (T7PO) enthält, X52, mit 0 initialisiert. Dann werden in einer Schleife alle Indizes in X50 von 0 bis CNT-1 (CNT steht in X51) durchlaufen.

Als nächstes wird die Stellenzahl aus T7TY und T8BY ermittelt. Dazu wird T7BY nach X54 geladen und entsprechend T7TY mit einem Faktor multipliziert:

T7TY	Faktor
C	1
P	2, vorher X54 um 1 (Längenbyte) erniedrigen
H	2
F	2

Ist T7TY mit keinem der oben aufgeführten Typen belegt, so wird Fehler #50.01 gemeldet. Ansonsten wird der errechnete Wert in T7ST+3∗X50 geschrieben.

Im Folgenden wird T6K1 nach T7K1 kopiert und falls das Erläuterungskennzeichen T7KZ nicht gesetzt (Blank) ist, so wird es auf "N" gesetzt.

Zum Schluß wird das Schlüsselfeld-Kennzeichen T7SF auf 'J' gesetzt, falls gilt: T6KP $\leq$ T7PO $<$ T6KP+T6KL

Register:

- X50... Indexzähler
- X51... höchster Index+2 (CNT)
- X52... Summe aller T7BY = Position im Satz
- X53... für Index-Offset-Umrechnungen
- X54... Zwischenspeicher für T7BY (für T7ST)
- X55... laufender Zähler (für T8K4)

SSUB10: Satzart ermitteln

Hier wird zuerst überprüft, ob TOKL3 ungleich 0 ist, d. h. ob eine Satzartennummer angegeben ist. Ist dies nicht der Fall, so wird die nächste (zweite) Macro-Zeile eingelesen. Falls diese ebenfalls keine Satzart-Bezeichnung enthält, wird Fehler #10.01 ausgegeben, da die Satzart-Nummer in der ersten oder zweiten Zeile stehen muß.

Ansonsten wird überprüft, ob der Dateiname, der in der ersten oder zweiten Zeile stehen muß und nun in TOK3 steht und die Satzart-Nummer enthält, mit dem Namen, über den das Macro z. B. im Editor angesprochen wird, identisch ist. Ist dem nicht so, wird dies als Fehler #10.03 gemeldet. Diese Abfrage ist dann interessant, wenn Macros versehentlich überschrieben wurden...

Register:

- X50... Quell-Zeiger für Zahlen, TOK3
- X51... Anzahl Zeichen in CFELD
- X52... Ziel-Zeiger in CFELD
- X53... Diverses
- X54... Länge des Macro-Namens=TOKL3

SSUB20: Macroname suchen

Dieses Unterprogramm liest aus der 2. Spalte (TOK2) der dritten Zeile den Macro-Namen. Wurde in SSUB10 die SA-Nummer bereits in der ersten Zeile gefunden, würde hier die zweite Zeile eingelesen werden. Dies kann daran festgestellt werden, daß in der zweiten Zeile das Schlüsselwort "`MACRO`" steht. Wird dieses in TOK2 gefunden, so wird eine Zeile weiter gelesen und der Name erst dann aus TOK2 nach T6MN kopiert. Außerdem wird die Länge des Macro-Namens, die in TOKL2 steht, nach T6MNL kopiert. Dieser Wert wird in SSUB30 gebraucht.

Register:

- X50... Diverses

SSUB30: Satzartenbezeichnung einlesen

SSUB30 liest die Satzartenbezeichnung aus der 2. Kommentarzeile, nachdem alle vorhergehenden Zeilen überlesen werden. Die 2. Kommentarzeile muß den folgenden Aufbau haben:

```
*  <beliebiger Text>  <Macro-Name>  <Satzarten-Bezeichnung>
```

Der Macroname muß mit dem Macronamen in T6MN aus Zeile 3 übereinstimmen. Von der Satzartenbeschreibung werden die ersten 25 Zeichen nach T6BE kopiert.

Wird der Macro-Name nicht gefunden, so wird ein Fehler #30.01 gemeldet.

Register:

- X50... Zeiger auf aktuelle Position in TOK4
- X51... Diverses

SSUB41: Filetable verarbeiten

Hier werden die Felder T6DA (Dateiname), T6TY (Zugriffsart: sequentiell oder indexsequentiell), T6SL (Satzlänge), T6KL (Schlüssellänge) und T6KP (Schlüsselposition) belegt, indem die in der aktuellen Zeile stehende Filetable, die durch das Macro DFFLAB gekennzeichnet ist, analysiert wird.

Die Argumentzeile des DFFLAB-Macros steht in TOK3. Diese Zeichenkette wird durchgelesen und bei bestimmten Schlüsselwörtern werden die entsprechenden Informationen gespeichert:

"`NAME=`": Die Zeichen bis zum nächsten Komma werden nach T6DA kopiert. Ist ein Dateiname länger als 8 Zeichen, so werden die restlichen max. acht Zeichen hinter dem Wortsymbol "`NAME2`" abgelegt.

“NAME2=”: Die folgenden Zeichen stellen den zweiten Teil des Dateinamens dar, mit Hilfe dessen auf die jeweilige Satzart zugegriffen wird. Er wird nach T6DA+8 kopiert.

“ORG=”: Das nächste Zeichen nach dem Gleichheitszeichen wird nach T6TY kopiert, da entweder “ISAM” oder “SAM” folgt.

“RECL=”: Die folgende Zahl gibt die jeweilige Satzlänge an und wird in T6SL angelegt.

“KEYL=”: Die Schlüssellänge, die diesem Wortsymbol folgt, wird nach T6KL kopiert.

“KEYP=”: Hierauf folgt die Schlüsselposition, mit der T6KP belegt wird.

Wurde eine Zahl oder ein Zeichen eingelesen, so wird der aktuelle Zeiger auf das erste Zeichen des nächsten Symboles positioniert, und falls dieses Zeichen ungleich Blank ist, wird dieses Symbol ebenfalls untersucht.

Register:

- X50... aktueller Zeiger in TOK3
- X51... Länge der verbleibenden Zeile

SSUB42: Satzartendefinition durchlesen

Diese Funktion liest mit Hilfe der zwei Funktionen ALYSE1 und ALYSE2 die Satzarten-Definition ein. ALYSE1 ist für die Datendeklaration (DC) zuständig, ALYSE2 für die Kommentare, siehe dort.

In X62 steht der Index für die Elemente der Feld-Tabelle, die als nächstes belegt werden, in X63 steht der entsprechende Index für die Erläuterungen. Da in Assembler keine komplexen Datenstrukturen möglich sind, wurden für die einzelnen Komponenten je eine eigene Tabelle angelegt, anstatt eine Tabelle, die die Struktur enthält, anzulegen.

Mit GETLN wird eine Zeile eingelesen. Beinhaltet sie eine “DC”-Anweisung sowie ein Label das auf SYMLEN Zeichen mit SYM übereinstimmt, so wird ALYSE1 aufgerufen. Ist zusätzlich ein Kommentar vorhanden, so wird außerdem ALYSE2 ausgeführt. Dann wird die nächste Zeile eingelesen. Ist diese eine reine Kommentarzeile (d. h. steht in der ersten Spalte ein Stern), so eird ALYSE2 erneut aufgerufen, die Daten werden noch für das Element der letzten “DC”-Zeile eingetragen. Zeilen, die weder “DC” enthalten, noch reine Kommentarzeilen sind, werden ignoriert, außer in TOK2 steht das Schlüsselwort “ORG”. Dies ist das Zeichen, daß keine weiteren Felder mehr folgen. Ansonsten wird der Index-Zähler X62 erhöht, die laufende Nummer der Erläuterungsfelder (X60) wird auf 1 zurückgesetzt und an den Anfang (GETLN) der Schleife gesprungen.

Wurde “ORG” gefunden, so wird der höchste Index+1 nach CNT geschrieben, er wird u. a. beim Durchlesen der Drucktabelle wieder gebraucht. Ebenso wird der Erläuterungsindex von X63 nach CNT2 kopiert.

Register:

- X60... laufende Nummer der Erläuterungs-Felder
- X61... Länge der Macro-Namens ohne Feldsymbol = SYMLEN
- X62... Index für Feld-Tabellen T7xx
- X63... Indef für Erläuterungen T8xx

ALYSE1: Datendefinition analysieren

Diese Funktion unterstützt das Erfassen der Satzarten-Definition. Sie liest das Symbol, Typ und Byteanzahl des jeweiligen Feldes ein. Beim Aufruf steht in X62 der Index der Elemente, die gesetzt werden. In X61 steht SYMLEN.

Zuerst wird das Symbol nach T7SY + 16 ∗ X62 und D7SY kopiert, um mit DPFTC/P Bildschirmausgaben machen zu können. Da bei den Macros DPFTP und DPFTC feste Adressen angegeben werden müssen, an denen die einzelnen Werte stehen, sind die zusätzlichen Felder D7.. nötig, da eine Referen zierung in den Tabellen nicht möglich ist. Das Symbol, das nach T7SY bzw. D7SY kopiert wird besteht aus TOKL1−SYMLEN Zeichen, die ab TOK1+SYMLEN stehen.

Als nächstes wird die Bytezahl ermittelt. Die Deklaration kann dabei die folgenden Formen annehmen:

`aLbt'..'` bzw. `Lbt'..'` bzw. `t'..'`

Dabei ist `a` ein Wiederholungsfaktor, `b` die Anzahl der zu reservierenden Bytes und `t` der Darstellungstyp. Die Typen `C`, `P`, `H` und `F` werden unterstützt. Bei der letzten Deklarationsform wird für `F`- und `P`-Felder eine Byte-Anzahl von 2, bei `C`- und `H`-Felder 1 Byte in T7BY+3∗X62 eingetragen. `a` und `b` müssen Zahlenwerte sein, ungültige Zeichen werden als Fehler #42.11 (Fehler bei `a`) und #42.12 (Fehler bei `b`) ausgegeben.

Der jeweilige Typ wird in T7TY abgelegt; er wird erst in DOREST, bei der Berechnung der Stellenzahl, auf Gültigkeit überprüft!

Desweiteren wird in ALYSE1 die Feldnummer T7K3 auf den Wert des aktuellen Index' +1 gesetzt. Damit ergeben sich die Feldnummern in der Reihenfolge, in der die Feldelemente definiert werden. Ist eine Drucktabelle vorhanden, werden die Werte gemäß dieser überschrieben.

Register:

- X50... Quelle: TOK2
- X51... Für Zeichenklassen-Bestimmung, siehe "`CCT`" auf Seite 14.

- X52... Ziel für `a` und `b`: CFELD
- X53... Zwischenspeicher für Byte-Länge `b`
- X54... Für Index-Offset-Umrechnungen
- X61... SYMLEN
- X62... aktueller Index

ALYSE2: Kommentar analysieren

Diese Funktion ist für das Verarbeiten der Kommentare in den Satzarten-Definitionen verantwortlich. Der Kommentar kann eine oder mehrere der folgenden Informationen enthalten, wobei mehrere Felder durch ein Komma getrennt werden müssen:

‘‘...‘‘+: Bezeichnung; diese kann doppelt vorkommen, die Bezeichnungen werden in T7B1 und T7B2 abgelegt. Maximal werden 25 Zeichen gespeichert, ansonsten wird mit Leerzeichen aufgefüllt. Wird kein zweites Anführungszeichen gefunden, so wird Fehler #42.23 ausgegeben.

NK=: Anzahl Nachkommastellen; dies werden in T7AN abgelegt.

G=: Gruppierungskennzeichen, wird in T7GK abgelegt.

E=: Erläuterungen; Ist dieses Feld vorhanden, so wird T7KZ auf “J” gesetzt und vom folgenden Erläuterungstext werden die ersten 15 Zeichen nach T8ER kopiert. Der Rest der Zeile wird überlesen, da keine Anführungszeichen eingeführt wurden und so nicht feststellbar ist, wo die Erläuterungen enden. Pro Satzarten-Feld dürfen mehrere Ë=Felder vorkommen, jede Erläuterung erhält einen eigenen Eintrag in T8xx.

***:** Tritt in einem Kommentar (außer dem Stern zu Zeilenbeginn bei reinen Kommentarzeilen) ein Stern auf, so wird dieser und der Rest der Zeile übergangen. Dies ermöglicht es, weitere Kommentare anzubringen.

Felder am Ende einer Zeile brauchen kein abschließendes Komma, da ihnen kein weiteres Feld folgt.

Wird beim durchlesen des Kommentars auf ein (doppeltes) Anführungszeichen gestoßen, so wird zuerst überprüft, ob T7B1 (Satzart-Bezeichnung 1) bereits belegt ist. Falls ja, wird dies für T7B2 überprüft. Ist dieses ebenfalls schon besetzt, d. h. werden mehr als zwei Bemerkungen angegeben, so wird ein Fehler #42.21 ausgegeben.

Wird ein Symbol gefunden, daß keinem der oben aufgeführten entspricht, so wird ein Fehler #42.22 ausgegeben.

Register:

- X50... Zeiger auf Quelle = TOK4
- X51... Diverses
- X52... TOKL4
- X53... Für Index-Offset-Umrechnungen
- X54... CE-Zwischenspeicher
- X60... lfd. Zähler
- X61... Länge des Macro-Namens ohne Feldsymbole=SYMLEN
- X62... aktueller Feld-Index
- X63... Erläuterungs-Index

SSUB43: SUB-Routinen überlesen

Dieses Unterprogramm überliest die in MS-Macros enthaltenen Unterprogramme. Beim Aufruf steht "`SUB`" in TOK2. Es werden alle Zeilen überlesen, die kein "`RET`" in TOK2 haben.

SSUB44: Drucktabelle durchlesen

Hier wird die Drucktabelle, die in SCANMOD durch "`<SYM>XXX`" erkannt wurde, verarbeitet, falls Satzart-Felder gefunden wurden (CNT$\neq$ 0). Dabei werden die einzelnen Symbole, die in der Drucktabelle stehen, in T7SY gesucht. Wird ein entsprechendes Symbol gefunden, so wird der Index dieses Symbols in X54 gespeichert. T7K3+4$*$X54 wird dann auf den Wert eines Zählers gesetzt, der anschließend um 1 erhöht wird. Wird ein Symbol aus der Drucktabelle nicht in T7SY gefunden, so wird es übergangen und kein Fehler ausgegeben. So können mit "`+`" referenzierte Feldelemente überlesen werden.

Ist auf diese Weise eine Zeile abgearbeitet, wird mit GETLN die nächste Zeile eingelesen und, falls sie kein "`ANOP`" enthält, wird mit ihr ebenso verfahren.

Enthält die zuletzt gelesene Zeile in TOK2 ein "`ANOP`", so gilt die Drucktabelle als beendet.

Register:

- X50... Quell-Pointer auf jeweiliges Symbol
- X51... SYMLEN
- X52... Zähler für Feldnumer T7K3
- X53... Index des Sybols, auf das X50 zeigt

- X55... TOKL3
- X57... Zwischenspeicher für Symbol-Länge (⇐ CE)
- X58... CNT (Höchster Index = Anzahl Elemente in T7)

SSUB45: Matchcodes ermitteln

Dieses Unterprogramm ist für die korrekte Erfassung der Matchcode-Satzarten verantwortlich. Nachdem in SCANMOD erkannt wurde, daß in TOK2 "`AIF`" steht und in TOK3 die Zeichenfolge "`MC`" mit anschließendem "`Y`" vorkomt, wird hier festgestellt, welcher Matchcode (MC1–MC6) mit welcher Satzart verknüpft wird. Um dies zu erreichen, wird zuerst TOK3+2 betrachtet und entsprechend der dortigen Zeichenkette wird dann — falls sie zwischen "`MC1`" und "`MC6`" liegt — die nächste Zeile eingelesen. In dieser Zeile steht in TOK3 ab Position 2 die (Matchcode-)Satzart, aus der die MC-Satznummer ermittelt werden kann. Das eigentliche Herauslesen der SA-Nummer aus der Satzart wird in GETMC gemacht. Indem dies in ein weiteres Unterprogramm gepackt wurde, wird vermieden, daß der selbe Code für jeden der sechs Matchcodes einzeln geschrieben werden muß.

Verwendete Register:

- Keine!

GETMC: Hilfsfunktion für SSUB45

Nachdem in SSUB45 die Zeile, in der die MC-Satzart steht, eingelesen wurde, wird hier die Satzarten-Nummer herausgelesen. Dazu wird die erste Ziffer der Satzart gesucht und von dieser Position ab alles bis zum abschließenden Hochkomma nach CFELD kopiert. Dort wird die Satzarten-Nummer in das P-Feld PFELD umgewandelt, wo es von SSUB45 zur weiteren Verarbeitung erwartet wird.

Register:

- X50... Zeiger auf Nummern-Beginn
- X51... Zeichenklasse des Zeichens, auf das X50 zeigt

4.4 Low-Level-Functions

GETLN: Logische Zeile einlesen

Diese Routine liest eine logische Zeile ein, d. h. es wird eine Zeile aus der Macro-Datei gelesen und, falls sie ein Fortsetzungszeichen (Zeichen "`F`" in Spalte 72) enthält, wird die

nächste Zeile ohne zwischenstehende Leerzeichen angehängt. Es dürfen mehrere Fortsetzungszeilen existieren.

Bevor irgend etwas eingelesen oder angehängt wird, werden zuerst die Register X40 bis einschließlich X59 im C-Feld STACK abgelegt. Da der CTM-Assembler über keinen Stack verfügt, muß der Umweg über den Speicher gegangen werden.

Die einzelnen Zeilen werden in GRAWLN (Get Raw Line) in ZEILE2 eingelesen, nach ZEILE kopiert und solange ein Fortsatzungszeichen existiert, wird in einer Schleife die nächste Zeile (in ZEILE2) eingelesen und direkt an ZEILE angehängt. Durch dieses Vorgehen kann die Filetable auf einmal verarbeitet werden. Dies ist im Übrigen auch die einzige Routine, in der eines der Macros für die strukturierte Programmierung verwendet wurde, das WHILE-Macro.

Im Struktogramm sieht dieser Ablauf folgendermaßen aus:

9cm8mm Nächste Zeile nach ZEILE2 holen (GRAWLN) 71 Zeichen von ZEILE2 nach ZEILE kopieren (evtl. vorhandenes Fortsatzungszeichen nicht mitkopieren!) Zeichen in Spalte 71 gleich 'F' DO Nächste Zeile lesen (GRAWLN) X40=Erstes Zeichen in ZEILE2 X41=Letztes Zeichen in ZEILE 70-X40 Zeichen ab ZEILE2+X40 nach ZEILE + X41 + 1 kopieren

Abschließend wird die ZEILE noch in CUTLN (Cut Line) in die 4 Token TOK1 bis TOK4 kopiert und die Register X40 bis X59 wieder zurückgeholt.

Register:

- X40... Index des ersten Zeichens in ZEILE2
- X41... Index des letzten Zeichens in ZEILE
- X42... Zählregister
- X43... Adress-Zeiger

GRAWLN: Zeile aus Macro-Bibliothek holen

Hier wird eine Zeile aus der Macro-Bibliothek in PRSOURCE eingelesen und die ersten 72 Zeichen (die Zeilen sind je 80 Spalten breit, in den Spalten 72-80 steht das Datum der letzten Änderung) in das C-Feld ZEILE2 kopiert, es wird mit Leerzeichen aufgefüllt. Außerdem wird das in den Spalten 73–80 enthaltenen Datum (Format: TT.MM.JJ) in der Form JJJJMMTT im P-Feld DATUM2 abgelegt und mit dem P-Feld DATUM verglichen. Ist DATUM2 größer (jünger) als DATUM, so wird DATUM der Wert von DATUM2 zugewiesen. Auf diese Weise steht in DATUM am Ende des Macros, wann es zuletzt verändert wurde.

Die Jahrtausend und -hundert-Stelle des Datums werden per Default auf “19” gesetzt, falls die Jahreszahl über “50” liegt, sonst auf “20”.

Desweiteren wird das P-Feld ZNR (Zeilennummer) pro gelesener Zeile um 1 erhöht. Dies ist beim Ausgeben von Fehlern (in SMERR) nützlich.

Außerdem werden die letzten drei Bildschirmzeilen je um 1 Zeile nach oben gerückt, die soeben gelesene Zeile wird zuunterst angehängt. Dadurch wird stets ein zusammenhängender Ausschnitt aus dem aktuellen Macro angezeigt.

Tritt beim Einlesen einer Zeile ein Dateiende auf, so wird die Funktion EOFEND angesprungen.

GRAWLN wird nur von GETLN aufgerufen!

Register:

- X40... Spaltenzähler

CUTLN: Zeile in 4 Tokens zerlegen

Hier werden die in GETLN eingelesenen (und evtl. zusammengefügten) Zeilen in die folgenden Eingabesymbole zerlegt und in TOK1–4 abgelegt. Die Längen der einzelnen Token stehen in TOKL1–4. Die vier Token sind im einzelnen:

1. Label: Dieses beginnt in der ersten Spalte mit einem Zeichen ungleich Space. Ist in der erste Spalte ein Leerzeichen, so ist TOK1 mit Leerzeichen aufgefüllt und TOKL1 ist mit 0 belegt. Ansonsten steht in TOK1 das jeweilige Label, mit Leerzeichen aufgefüllt, in TOKL1 steht die Anzahl der Zeichen in TOK1.
2. Assemblerbefehl: Durch mindestens ein Leerzeichen getrennt kommt als nächstes ein Assemblerbefehl, der in TOK2 eingelesen wird. Die Länge des Befehls steht in TOKL2.
3. Argumente: Wiederum dirch mindestens ein Leerzeichen getrennt können nun die Parameter des Assemblerbefehls folgen, sie sind je nach Befehl optional, dürfen aber keine Leerzeichen enthalten (Ausnahmen siehe GTOK)! Die Argumente werden zusammen in TOK3, ihre Länge in TOKL3 abgelegt.
4. TOK4 wird mit dem folgenden, erneut durch mindestens ein Leerzeichen getrennten Kommentar gefüllt. Ist kein Kommentar vorhanden, so ist TOK4 mit Leerzeichen gefüllt, TOKL4 ist 0.

   Besteht eine Zeile nur aus einem Kommentar, d. h. steht in der erste Spalte ein '*', so sind alle Felder leer bzw. auf 0 gesetzt, nur TOK4 und TOKL4 sind belegt.

Das Einlesen der jeweiligen Symbole sowie von Kommentaren und das überlesen trennender Leerzeichen wird von den Funktionen GTOK (Get Token), SKPBLK (Skip Blank) und GETKMT (Get Kommentar) erledigt.

Register:

- X40... Spaltenzähler
- X41... Index des letzten Zeichens in ZEILE
- X43... Zeiger auf ZEILE
- X50... wird überschrieben
- X51... wird überschrieben
- X52... wird überschrieben

GTOK: Ein Token einlesen

GTOK wird in X50 die Adresse des Feldes übergeben, in die das Token geschrieben werden soll, auf das X52 zeigt. In X41 steht die Adresse des letzten Zeichens der Zeile. Die Aufgabe von GTOK ist es nun, das Eingabesymbol (Token) zu kopieren. Ein Token wird durch Leerzeichen begrenzt, da aber z. B. im Argument des "`AIF`"-Befehls Leerzeichen vorkommen, wird auf Leerzeichen in Klammern und (einfache) Anführungszeichen Rücksicht genommen. Steht vor einem öffnenden Anführungszeichen ein "`L`", so wird es ignoriert, um Längenbestimmungen der Form "`L'SA...`" zuzulassen.

Falls am Ende der Eingabezeile (Index des letzte Zeichens steht in X41) der Klammernzähler X53 oder der Anführungszeichen-Schalter X54 nicht auf 0 stehen, wird ein Fehler 02.01 ausgegeben.

Am Ende der Routine zeigt X52 auf das Leerzeichen nach dem Eingabesymbol.

Register:

- X50... Adresse des (Ziel-)Token-Feldes
- X51... Übergabe: Zeiger auf Token; Rückgabe: Anzahl gelesener Zeichen = TOKL
- X52... Rückgabe: Zeiger auf Blank nach Token
- X53... Level-Zähler für Klammerung
- X54... 0/1-Switch für Anführungszeichen; 1=in Anführungszeichen

SKPBLK: Bis zum nächsten Zeichen lesen

Dieser Routine wird ein Zeiger auf ein oder mehrere Leerzeichen übergeben, sie überliest diese Leerzeichen und gibt einen Zeiger auf das erste Nicht-Leerzeichen sowie die Anzahl der überlesenen Zeichen zurück.

Register:

- X50... Übergabe: Zeiger auf 1. Leerzeichen; Rückgabe: Zeiger auf 1. Nicht-Leerzeichen
- X51... Anzahl überlesener Zeichen

GETKMT: Kommentar am Zeilenende einlesen

Diese Routine ist ähnlich zu GTOK, sie stellt jedoch alle Zeichen ab (X52) bis zum Zeilenende im C-Feld, dessen Adresse in X50 übergeben wird, ab, ohne auf Klammern oder Leerzeichen zu achten. Die Anzahl der einzulesenden Zeichen wird in X51 übergeben.

Diese Funktion besteht nur aus einem MVC-Befehl:

```
GETKMT   SUB
         MVC  (X50),(X52),(X51)
         RET  GETKMT
```

Register:

- X50... Adresse von TOK4 (Ziel)
- X51... Anzahl zu kopierender Zeichen
- X52... Zeiger auf Kommentar-Anfang (Quelle)

SMERR: Zentrale Routine zur Fehlerausgabe

SMERR wurde implementiert, um einheitliche Fehlermeldungen zu erhalten. Sie werden jeweils auf dem Drucker ausgegeben bzw in der 2. Bildschirmzeile angezeigt und haben folgendes Format:

```
aaaaaaa: Fehler bb.bb in Zeile cccc: dddd....
```

Dabei ist 'a' der Name des aktuellen Macros, er steht in PRLABEL+1. Zwar ist er auch in T6MN zu finden, aber im Falle des Fehlers #10.03 (siehe Punkt A) ist PRLABEL+1 günstiger. 'b' entspricht der Fehlernummer, die in X50 übergeben wird. Ein Verzeichnis aller Fehlermeldungen befindet sich auf Seite 47.

'c' ist die Nummer der zuletzt gelesenen Zeile, als der Fehler erkannt wurde; sie wird aus ZNR genommen. 'd' ist der Kontext, in dem der Fehler steht; ein Zeiger auf ihn wird in X51 übergeben, in X52 steht die Anzahl der auszugebenden Zeichen.

Die Fehlermeldung wird im Puffer ERRBUF zusammengebaut und dann vom PRINT-Macro ausgedruckt, sofern FLAGPR mit "J" belegt ist.

Im weiteren wird dann ein Verweis auf das nächste Macro geholt und dieses verarbeitet, oder, falls der Fehler im letzten Macro aufgetreten ist, das Programm ordnungsgemäß beendet.

Register:

- X50... Fehlercode
- X51... Zeiger auf Fehler-Kontext
- X52... Kontext-Länge

CLRALL: Alle Datenfelder löschen

Hier werden zuerst alle Flags und sonstigen P-Felder gelöscht. Als nächstes wird der Zwischenspeicher für den Satz S9006 (T6..) gelöscht. Dann werden die Tabellen gelöscht, in denen die Informationen über die einzelnen Satzartenfelder (T7..) und Sartarten-Erläuterungen (T8..) enthalten sind. Da jede Tabelle 150 Elemente besitzt, müssen für jede Satzart zwei Schleifen 150 mal durchlaufen werden: einmal, um die P-Felder zu löschen, zum anderen, in die C-Felder mit Leerzeichen aufzufüllen. Außerdem werden die Felder, in denen die jeweils aktuellen Werde der Tabellen T7.. und T8.. für die Bildschirmausgabe zwischengespeichert werden (D7.. und D8..) initialisiert.

Folgende Register werden in CLRALL verwendet:

- X40... Zähler
- X41-47... Zeiger auf zu löschende Felder

5 Bedienung der einzelnen Programme

Bevor nun ein Dokumentierter Testlauf erfolgt, soll hier kurz die Bedienung der einzelnen Programme erläutert werden.

5.1 S-Programm: S9006

Dieses S-Programm erlaubt es, Daten über die Satzarten-Köpfe abzurufen. Der Aufruf erfolgt aus dem Verteiler "`SML - SATZARTEN`" über Funktion 9006#.

Um Informationen abzurufen, ist die Eingabe der Satzartennummer (auf der numerischen Tastatur, mit Plus-Taste abgeschlossen) und eines Versionskennzeichens erforderlich. Wurde der Schlüssel mit den notwendigen Daten gefüllt und die Eingabe mit der Stern-Taste abgeschlossen, so können die gewünschten Werte betrachtet werden. Wird bei der Eingabe des Schlüssels Funktion 55# angewählt, so wird der erste Satz angezeigt.

Beim Betrachten der Werte sind folgende Funktionen möglich:

55#: Folgesatz; damit kann der nächste Satz (alphabetisch geordnet) angezeigt werden.

32#: Satz ändern. Nach dieser Funktion können die folgenden Funktionen aufgerufen werden:

44#: "Satz konstant" setzen

46#: "Satz konstant" löschen

39#: "Satz löschen", es erfolgt keine weitere Rückfrage!

Nach anwählen von Funktion 32# können die einzelnen Satzart-Felder mit den Cursortasten angewählt und manuall verändert werden.

5.2 S-Programm: S9007

Mit Hilfe dieses Programms können Informationen über einzelne Felder einer bestimmten Satzart erhalten werden. Es wird ebenfalls, wie S9006, aus "`SML - SATZARTEN`" aufgerufen. Die Nummer des Funktionsaufrufes ist 9007#, die weitere Bedienung ist die selbe wie S9006.

5.3 S-Programm: S9008

Ist in einem Feld aus S9007 "`Kennzeichen Erläuterungstext`" mit einem "`J`" (Ja) gekennzeichnet, so existiert zum jeweiligen Feld eine Erläuterung. Diese Erläuterung kann hier abgefragt werden, indem man die Satzart, ihre Versionsnummer und die Feldnummer des mit '`J`' gekennzeichneten Feldes angibt oder mit Funktion 55# durchblättert. Die weitere Bedienung ist analog zu S9006 und S9007. Wie auch die vorhergehenden S-Programme wird aus dieses aus "`SML - SATZARTEN`" aufgerufen, hier mit Funktion 9008#.

5.4 Hauptprogramm: DSS04

Dieses Programm erstellt das eigentliche Data Dictionary, indem es die vorgegebenen MS-Macros durchliest und die gewonnen Daten in den Satzarten S9006–S9008 abstellt, die dann mit Hilfe der oben beschriebenen Satzartenprogramme betrachtet werden können. Es werden folgende Eingabe gefordert:

Bibliotheks-Nummer: Nummer der Bibliothek, in der die durchzulesenden MS-Macros abgespeichert sind, im Normalfall `1`.

Von Macro: Name des ersten Macros, das eingelesen werden soll.

Bis Macro: Name des letzten Macros, das erfaßt werden soll. Die Einstellung dieses Feldes wird von "Von Macro" übernummen und kann bei Bedarf, d.h. falls mehrere Macros in die Datenbank aufgenommen werden sollen, geändert werden.

Protokolldruck: Wird hier ein "`J`" (Ja) eingegben, so werden die Macros im Batchbetrieb erfaßt, nach Bearbeitung eines Macros wird eine Meldung auf den Drucker ausgegeben. Wird "`N`" (Nein) eingegeben, so werden die jeweiligen Ausgaben nur auf den Bildschirm ausgegeben und anschließend auf die *Stern*-Taste gewartet.

Die Eingabe dieser Werte wird mit der Stern-Taste abgeschlossen. Daraufhin werden die Macros der Reihe nach durchgelesen und die gewonnenen Daten am Ende jedes Macros in die Datenbank geschrieben.

Treten Fehler auf, d.h. erfüllt ein Macro nicht das geforderte Format (siehe Punkt 6 auf Seite 35), so wird eine Fehlermeldung (siehe Punkt A, Seite 47) auf den Drucker oder den Bildschirm ausgegebe, entsprechend der Wahl bei *Protokolldruck*, ausgegeben, und beim nächsten Macro weitergemacht. Bei *Protokolldruck=Ja* können so mehrere Macros als Batch-Job erfaßt werden, da vom Benutzer keine weiteren Eingaben erforderlich sind. Es ist lediglich am Ende des Jobs zu prüfen, ob Fehler aufgetreten sind: diese werden am Drucker protokolliert. Sie müssen korrigiert werden, da über die fehlerhaften Macros noch keine Informationen im Data Dictionary stehen.

Wenn Fehler auftreten und der Protokolldruck nicht eingeschaltet ist, so werden in der ersten Zeile nähere Informationen geliefert. Diese Zeile enthält die Meldung "<Macroname> `ok.`", falls keine Fehler aufgetreten sind. In jedem Fall kann durch einen Druck auf die *Stern*-Taste mit der Verarbeitung fortgefahren werden.

6 Aufbau eines MS-Macros

Im Folgenden werden die Teile der MS-Macros beschrieben, die beim Einlesen durch DSS04 von Bedeutung sind bzw. worauf zu achten ist.

In der ersten Zeile steht nach `MODBEG` der Macro-Name, aus dem die Satzart bestimmt wird. Dieser Name kann auch wahlweise in der zweiten Zeile (nach `MACRO`) stehen.

In der dritten Zeile steht der Macro-Name, der auch im Data Dictionary erscheint. In der selben Zeile werden diverse Assembler-Konstanten gesetzt, diese sind aber für das korrekte Einlesen nicht notwendig.

Als nächstes folgt ein Block von Kommentarzeilen. In der zweiten Kommentarzeile muß der Macro-Name aus Zeile 3, gefolgt von der Satzart-Bezeichnung, stehen.

Dann empfiehlt es sich, die Filetable (falls benötigt, s. u.) zu definieren. Beim Macro `DFFLAB` müssen die folgenden Felder belegt sein: `NAME`, evtl. `NAME2`, `ORG`, `RECL`, `KEYL` und `KEYP`. In Satzarten, die keine Filetable enthalten oder benötigen, werden diese Informationen aus der ersten "`DC`"-Anweisung des MS-Macros gelesen. In ihr steht die Adresse

einer (externen) DFFLAB. Zur Zeit sind die DFFLABs "TIMOS", "T102" und "TIMST" bekannt.

Die Satzarten-Definition wird an einem Label, gefolgt von einem "DS" erkannt. Ist das erste Zeichen des "DS"-Argumentes keine "0", so wird davon ausgegangen, daß nur Speicher reserviert wird und keine Satzart-Felder definiert werden. Folgt der "DS"-Anweisung eine Null, so werden die folgenden Felder bis zum nächsten "ORG"-Befehl erfaßt.

Die einzelnen Felder der Satzarten-Definition haben das folgende Format:

```
<Label wie bei DS><Feldsymbol>   DC   aLbt'...'   Kommentar
```

Felder ohne Label oder mit Labels, die nicht so beginnen, wie das Label der DS-Zeile sowie Felder, die andere Befehle als "DC" enthalten, werden überlesen! Bei der eigentlichen Datendeklaration bedeutet 'a' einen Wiederholungsfaktor, 'b' die zu reservierende Byteanzahl und 't' den Feldtyp. Für 'a' und 'b' dürfen nur Zahlen, keine Ausdrücke eingesetzt werden. Bei b=1 kann b entfallen: "1L5..." entspricht "L5...". 'b' darf nicht fehlen (Assembler-Fehler!). Definitionen der Form

```
 t'...'
```

sind erlaubt, wobei 't' wiederum einer der Typen F, P, C oder H ist. Bei der letzten Form werden bei t=C oder t=H 1 Byte, bei t=F oder t=P 2 Bytes angenommen; der nachfolgende Text in Hochkomma wird nicht gelesen!

Der Kommentar kann weitere Informationen über das jeweilige Feld enthalten. Besteht eine Zeile nur aus Kommentar (Stern in erster Spalte), so werden die darin enthaltenen Informationen dem vorherigen Feld zugeordnet. Im einzelnen sind folgende Schlüsselworte erlaubt:

''...''+: Bezeichnungen; Diese müssen in Anführungszeichen eingeschlossen werden, es sind maximal zwei 25 Zeichen lange Kommentare erlaubt. Pro Feld sollte mindestens eine Bezeichnung angegeben werden.

NK=: Direkt hinter dem "=" folgt die Anzahl der Nachkommastellen des jeweiligen Feldes.

G=: Hier folgt dem "=" eine Zahl, die die Gruppierung beschreibt.

E=: Erläuterungen; Hier werden die ersten 15 Zeichen übernommen, Anführungszeichen sind nicht nötig, da das Erläuterungs-Feld das letzte im Kommentar sind muß, weitere Schlüsselworte nach dem "E=..." werden überlesen! Es dürfen mehrere Erläuterungen angegeben werden, diese erhalten je einen eigenen Eintrag in T8xx und können aufgrund der laufenden Nummer T8K4 unterschieden werden.

***:** Tritt in einem Kommentar (außer dem Stern zu Zeilenbeginn bei reinen Kommentarzeilen) ein (weiterer) Stern auf, so wird dieser und der Rest der Zeile übergangen. So können beliebig weitere Kommentare im Macro stehen.

Eine weitere Informationsquelle, die allerdings nicht unbedingt nötig ist, ist die Drucktabelle. Mit ihrer Hilfe kann die Feldnummer bestimmt werden. Ist keine Drucktabelle angegeben, so werden die Feldelemente in der Reihenfolge ihres Erscheinens in der Satzarten-Definition durchnumeriert.

Die Drucktabelle wird am Label erkannt, dessen letzte drei Buchstaben drei "`X`" sein müssen, z. B. "`S9008XXX`". Es werden alle folgende Zeilen eingelesen und verarbeitet, bis eine Zeile das Schlüsselwort " `ANOP`" enthält; es ist also nötig, Drucktabellen mit "`AIF...ANOP`" zu klammern!

In der Drucktabelle selbst stehen die Adressen der Felder aus der Satzart des jeweiligen Macros. Feld-Symbole, die nicht in der Satzarten-Definition vorkommen, werden übergangen. So ist es möglich, Felder, die über andere Feldnamen angesprochen werden, in der Drucktabelle zu lassen, z. B. "`Sxxxx+10`".

Der letzte Punkt, der beim Schreiben von MS-Macros zu beachten ist, betrifft die Matchcodes. Sie werden an entsprechenden "`AIF`"-Abfragen erkannt, müssen also ebenfalls mit "`AIF...ANOP`" geklammert werden. Beispiel:

```
          AIF      (&MC1 NE 'Y')_L100
          MS313M1
_L100     ANOP
```

Es werden nur die Matchcode-Satzarten erfaßt, die auf '`Y`' abgefragt werden, eine Abfrage auf `EQ` ist ohnehin nicht erlaubt (⇒ Transformator)!

Die soeben beschriebenen Punkte sollen am folgenden Beispiel (`M9008`) nochmals im Zusammenhang aufgezeigt werden. Die relevanten Stellen wurde unterstrichen:

```
          MODBEG
          MACRO     M9008
          MS9008    &DRTAB=N,&FTABLE=N
************************************************************
*         MACRO SATZAUFBAU M9008 Satzarten-Erlaeuterung
************************************************************
*
T908      DFFTAB    LABEL=L908,DEVICE=63,                    F
                    ACCESS=ISAM,COMMON=YES,                  F
                    ERRADR=DFERR,                            F
                    EOFADDR=DFERR,IOBUF1=IMOSBF
*
L908      DFFLAB    NAME=IMOS0000,NAME2=DSSSAERL,            F
                    ORG=ISAM,PROT=NO,RES=5000,               F
                    RECL=28,KEYL=10,KEYP=0
*
*
```

```
            DC          A'T908'
            DC          H'4,4'
            DC          F'0'
*
S9008       DS          0L32
*
S9008K1     DC          L3P'0'          "Satzart"
S9008K2     DC          L1C' '          "Versionskennzeichen"
*                                       E=' '=akt.,A=alt
S9008K3     DC          L3P'0'          "Feldnummer"
S9008K4     DC          L3P'0'          "lfd. Nummer"
S9008ER     DC          15L1C' '        "Erlaeuterungen"
            ORG         S9008+32
*
YP9008      DC          P'9008'
YC9008N     DC          C'N'
*
S9008LO     SUB
            ZP          S9008k1
            MVC         S9008K2,BLANK,1
            ZPM         (S9008K3,S9008K4)
            MVC         S9008BLANK,1
            MVC         S9008ER+1,S9008ER,14
            RET         S9008LO
*
            AIF         (&DRTAB NE 'J')_L100
S9008XXX    DC          A'S9008K1,S9008K2,S9008K3'
            DC          A'S9008K4,S9008ER'
*
_L100       ANOP
            MEND
            MODEND
```

7 Dokumentierter Testlauf

Der folgende Testlauf beschreibt die Erfassung des oben abgedruckten Macros M9008, das für die Verwaltung der Satzart-Erläuterungen zuständig ist.

Es wurde nur das Macro M9008 aus Bibliothek 1 erfaßt, Protokolldruck wurde auf 'J' gesetzt. Die folgende Abbildung zeigt das den Macro-Scanner DSS04 (siehe erste Bild-

schirmzeile!) in Aktion:

Der Bildschirm ist dabei in zwei Teile aufgeteilt. Die unteren 4 Zeilen zeigen den Ausschnitt des Macros, der gerade gelesen wird.

Der obere Teil zeigt die aktuellen Inhalte der Satzarten S9006 (links, 6K1 bis 6M6), Satzart9007 (mitte, 7SY bis 7GK) und Satzart9008 (rechts, 8K1 bis 8ER). Diese werden stets angezeigt, bevor eine neue Zeile eingelesen (und in der letzten Bildschirmzeile angezeigt) wird. Dies erklärt auch, warum im Feld 7F1 noch `Versionskennzeichen` steht, während in der letzten Zeile bereits `Feldnummer` angezeigt wird: die `Feldnummer`-Zeile wurde zwar eingelesen (und gleichzeitig angezeigt, siehe Funktion GRAWLN auf Seite 29), aber noch nicht verarbeitet.

Die Verarbeitung wird jedoch nicht — wir erwartet — ordnungsgemäß beendet, vielmehr wird folgendes auf den Drucker ausgegeben:

```
M9008 : Fehler 30.01 in Zeile 0003: * MACRO SATZAUFBAU M9008 Satz...
————————————————————————M9008–M9008—————-
```

Fehler 30.01 bedeutet, daß der Macro-Name vor Satzart-Beschreibung (zweite Kommentarzeile) nicht gefunden wurde (siehe Liste der Fehlermeldungen im Anhang). Der

Macro-Name — er wird aus der 3. Zeile gelesen — ist “MS9008”. Dies ist falsch, da die Satzart bereits 4 Stellen hat ist der richtige Name “M9008”. Nachdem die 3. Zeile also von

```
MS9008          &DRTAB=N,&FTABLE=N
```

in

```
M9008           &DRTAB=N,&FTABLE=N
```

angeändert und das Macro erneut erfaß wurde, meldet sich der Drucker mit:

```
M9008 : ok.
————————————————————————M9008–M9008————————-
```

Dies ist das Zeichen, das das Macro korrekt erfaß wurde und die darin enthaltenen Informationen nun im Data Dictionary stehen.

Mit Hilfe der einzelnun Satzarten-Programme können nun die abgestellten Informationen betrachtet werden.

S9006 liefert allgemeine Informationen über die im Macro M9008 definierte Satzart 9008:

Mit Hilfe des S-Programmes S9007 können nun die Daten der einzelnen Felder 1–5 abgefragt werden.

Feld 1: Satzart

Feld 2: Versionskennzeichen

Feld 3: Feldnummer

Feld 4: Laufende Nummer

Feld 5: Erläuterungen

A Fehlercodes des Macro-Scanners DSS04

01.01 ... Unbekannte Satzart in 1. DC-Anweisung; DFFLAB überprüfen: in der 1. DC-Zeile sind z. Zt. nur "`TIMOS`", "`T102`" und "`TIMST`" bekannt!

02.01 ... Klammern oder Anführugszeichen nicht ausbalanciert

04.02 ... Drucktabelle vor Satzarten-Definition gefunden!

05.01 ... Satzarten-Definition fehlen

05.02 ... keine DFFLAB oder DC gefunden

10.01 ... SA wird in 1. oder 2. Zeile erwartet

10.02 ... keine SA-*Nummer* in 1. oder 2. Zeile

10.03 ... Filename nicht gleich Macro-Name in erster Zeile; Wurde das Macro versehentlich überschrieben???

30.01 ... Macro-Name vor SA-Beschreibung nicht gefunden (zweite Kommentarzeile)

42.11 ... Fehler in Speicherdeklaration (SA-Definition): Es sind nur Zahlen, keine Ausdrücke erlaubt!

42.12 ... Fehler in Speicherdeklaration: Zahl nach '`L`' erwartet, kein Ausdruck

42.21 ... Nur zwei 25 Zeichen lange Kommentarfelder pro SA-Feld erlaubt

42.22 ... Unbekanntes Schlüsselwort in Satzart-Feld-Kommentar; erlaubt: `E=`, `NK=`, `G=`, `"..."` (2×), `*`

42.23 ... Offene Anführungszeichen entdeckt

50.01 ... Unbekannter Feldtyp in SA-Definition; Erlaubt: `P`, `C`, `H`, `F`

B Abkürzungsverzeichis

ABI ..... *A*pplication *B*inary *I*nterface

AIX ..... *A*dvanced *I*nteractive E*x*ecutive, IBM-eigene UNIX-Version, entspricht AT&T-System V Release 4

BAP ..... *B*ildschirm*a*rbeits*p*latz

BCD ..... *B*inary *C*oded *D*ecimal

BSD ..... *B*erkeley *S*ystem *D*istribution; neben AT&T-System V bedeutendste UNIX-Strömung

CASE ..... *C*omputer *A*dded *S*oftware-*E*ngineering, *C*omputer-*A*dded *S*ystem-*E*ngineering oder *C*omputer-integrated *A*pplication-*S*ystem-*E*ngineering, je nach Anwendungsgebiet

DNÜ ..... *D*aten*n*ah*ü*bertragung, lokales Netzwerk

ITOS ..... *I*ntelligent *T*erminal *O*perating *S*ystem, CTM-Betriebssystem

KSO ..... *K*unden–*S*ervice–*O*rganisation, Abteilung der UBG

LSB ..... *L*ease *S*ignificant *B*it, niedigstes Bit eines Bytes/Wortes

MFLOPS ..... *M*illion *FL*oating-point-*O*perations *P*er *S*econd)

MIPS ..... *M*illion *I*nstructions *P*er *S*econd

POWER ..... *P*erformance *O*ptimized *W*ith *E*nchanted *R*ISC, von IBM entwickelte Prozessor-Architektur

PPS ..... *P*roduktions*p*lanung und -*s*teuerung

RISC ..... *R*educed *I*nstruction *S*et *C*omputer; Spezielle IBM-Definition für den POWER-Prozessor: *R*educed *I*nstruction *S*et *C*ycles

S-Programm ... Satzarten-Programm, wird verwendet um Datensätze anzusehen oder zu erfassen

SCO ..... *S*anta *C*ruz *O*peration, Herstellers einer zu AT&T-SVR4-kompatiblen UNIX-Version

SVR4 ..... *S*ystem *V* *R*elease *4*, letzte Version des AT&T-UNIX', dürfte "das" UNIX werden

C Literaturverzeichnis

[1] Eva Kachl: Rückmeldung, *UNIX-Magazin* 9/91, Seite 16–24. Bericht über IBM-RS/6000-System mit Details über POWER-Prozessor.

[2] Unternehmensberatung Gommel GmbH, Weißenburgstraße 3, 8400 Regensburg: *UBG-Software-Richtlinien Band 10, Kapitel 2: Ablauf des Software-Erzeugungsprozesses*, Oktober 1991.

[3] Unternehmensberatung Gommel GmbH, Weißenburgstraße 3, 8400 Regensburg: *UBG-Software-Richtlinien Band 20: Systeminformationen*, Oktober 1991.

[4] Gottfried Vossen: *Datenmodelle, Datenbanksprachen und Datenbank-Managementsysteme*. Addison-Wesley, Bonn 1987.