TELEFUNKEN
COMPUTER

TAS
Telefunken-Assembler-Sprache

SYSTEM TR 440

Lfd. Nr.	Datum	Abschnitt	Seite	Berichtigung
1	Nov. 72	Der beiliegende Änderungsdienst 1 von Nov. 72 stellt eine vollständige Neuauflage der "TR 440 TAS-Telefunken-Assembler-Sprache" dar. Alle Ausgaben der bisherigen TAS-Sprache werden damit ungültig.		

SYSTEM TR 440

TAS
Telefunken-Assembler-Sprache

Einleitung A

Aufbau der TAS-Sprache B

Steuerung des Übersetzungsvorganges C

Die TAS-Makrosprache D

UEBERSETZE-Kommando E

Das TAS-Protokoll F

Standardrahmen für TAS-Programme G

Testhilfen H

Makrobibliotheksorganisation I

Große Befehlsliste J

Stichwortverzeichnis K

EINLEITUNG

Programme können entweder in einer problemorientierten Sprache oder in einer maschinenorientierten Sprache formuliert werden.
Problemorientierte Sprachen, z.B. ALGOL, PL/1, FORTRAN, COBOL wurden entwickelt, um bestimmte Arten von Aufgaben in einer bequemen, den Aufgaben entsprechenden Form darstellen zu können. Sie sind unabhängig von einer bestimmten Rechenanlage. Maschinenorientierte Sprachen sind auf einen bestimmten Rechner zugeschnitten. Sie setzen einerseits die Kenntnis der Maschinenbefehle voraus, ermöglichen aber andererseits eine optimale Formulierung aller auf dem Rechner überhaupt lösbaren Aufgaben.
Programme in der vom Programmierer geschriebenen Form heißen Quellenprogramme. Sie müssen erst durch einen Übersetzungsvorgang in eine für den Rechner verständliche Form gebracht werden. Ein Programm in dieser Form heißt Maschinenprogramm. Übersetzer für maschinenorientierte Sprachen werden Assembler genannt (Übersetzer für problemorientierte Sprachen heißen Compiler). Im Falle des TR 440 erzeugt der Assembler nicht unmittelbar Maschinenprogramme, sondern eine Zwischenform, den sogenannten Montagecode.
Aus dem Montagecode, der aus einem oder mehreren Quellenprogrammen gewonnen worden ist, wird von einem sprachunabhängigen Übersetzer, dem Montierer, ein gemeinsames Maschinenprogramm erzeugt. Die Quellenprogramme können dabei in verschiedenen Programmiersprachen geschrieben sein.

Alle neueren Programmiersprachen gestatten die Verwendung von Namen zur Bezeichnung von Adressen oder anderen maschineninternen Objekten. Sprachen, in denen Namen Objekte symbolisieren, werden symbolische Programmiersprachen genannt.
TAS (Telefunken-Assembler-Sprache) ist die maschinenorientierte symbolische Programmiersprache des TR 440.

Der Teil B der vorliegenden Beschreibung behandelt den Aufbau der TAS-Sprache (die Syntax).
Im Teil C werden die in der TAS-Sprache enthaltenen Steueranweisungen beschrieben, mit deren Hilfe der Ablauf des Übersetzungsvorgangs nach Bedarf beeinflußt werden kann. So können z.B. Teile des übersetzten Programms in schreibgeschützte oder nichtschreibgeschützte Teile des Kernspeichers gelegt werden.

Der Geltungsbereich bestimmter Namen kann eingeschränkt werden, wodurch größere Freiheit in der Namensgebung ermöglicht wird. Außerdem können Querbezüge zwischen einzelnen getrennt übersetzten Quellenprogrammen vereinbart werden.

Im Teil D wird die TAS-Makrosprache beschrieben. Sie ermöglicht u. a. eine Erweiterung der TAS-Sprache um neue Elemente, die sogenannten Makros. Dies sind Zusammenfassungen von TAS-Elementen. Diese werden vom Programmierer definiert oder liegen in einer Makrobibliothek vor und können danach in beliebiger Weise verwendet werden.

Weitere Kapitel befassen sich mit der Anwendung von Kommandos für TAS-Übersetzungen, mit dem vom Assembler erzeugten Protokoll und mit Hilfen für die Programmierung in TAS (Standardrahmen, Überwachung, Makrobibliothek).

Die vorliegende Schrift ist weitgehend für ein Lesen von vorn nach hinten aufgebaut. Weil sie jedoch auch zum Nachschlagen geeignet sein soll, wurde das Prinzip des didaktischen Aufbaues zugunsten einer inhaltlichen Gliederung bisweilen durchbrochen. In diesem Zusammenhang sei besonders auf das Kapitel C, Abschnitte 1 und 2 hingewiesen, in denen Bemerkungen von allgemeiner Bedeutung stehen.

AUFBAU DER TAS-SPRACHE

B

1.	Darstellung der TAS-Sprache	1
2.	Zeichenvorrat der TAS-Sprache	2
3.	Formatfreiheit	2
4.	Elemente der TAS-Sprache	3
4.1.	Namen	3
4.2.	Sondernamen	3
4.3.	Selbstdefinierende Ausdrücke	4
4.4.	Zahlen	4
4.5.	Tetradenfolge	4
4.6.	Oktadenfolge	5
4.7.	Textfolge	6
5.	Informationseinheiten	7
5.1.	Aufbau der Informationseinheiten	7
5.2.	Benennungsteil	8
5.3.	Informationsteil	8
5.4.	Spezifikationsteil	8
5.5.	Abgrenzungsteil	8
6.	IE-Spezifikationen	9
6.1.	Ablagespezifikationen (K, V, B, D)	9
6.2.	Typenkennungsspezifikationen (0, 1, 2, 3)	9
6.3.	Gerade- und Ungeradespezifikationen (G, U)	9
6.4.	Adressenkonstantenspezifikationen (A)	10
6.5.	Markierungsspezifikation (M)	10
6.6.	Zweitmarkierungsspezifikation (N)	10
6.7.	Rechts- und Linksspezifikation (R, L)	10
6.8.	Halbwortspezifikation (H)	10
7.	Befehle	11
7.1.	Befehlscode	11
7.2.	Adressenteile	12
7.3.	Ganzadresse	13
7.4.	Absolutbezüge	14
7.5.	Kernspeicherbezüge	14
7.6.	Links- und Rechtsadressen	15
7.7.	Indexbezug	16

8.	Literale	16
8.1.	Ablage bei Literalen	17
8.2.	Ganzwortliterale	18
9.	Pseudobefehle	18
10.	Konstanten	20
10.1.	Festpunktkonstante	21
10.2.	Gleitpunktkonstante	22
10.3.	Gleitpunktkonstante doppelter Genauigkeit	22
10.4.	Tetradenkonstante	23
10.5.	Oktadenkonstante	24
10.6.	Textkonstante	25
10.7.	Adressenkonstante	25
10.8.	Bitfeldkonstante	26
10.9.	Oktadenadressierung und OA-Befehl	27
10.10.	EA-Befehlswort	28
11.	Kommentare und Überschrift	28

1. Darstellung der TAS-Sprache

Die Struktur der Elemente der TAS-Sprache wird in diesem Handbuch nicht nur in Worten beschrieben, sondern zusätzlich auch in einer sogenannten Meta-Sprache dargestellt. Diese Darstellungsform ist in manchen Fällen anschaulicher und übersichtlicher als die wörtliche Beschreibung. In der Meta-Sprache haben folgende Zeichen eine besondere Bedeutung.

: : = entspricht; dient zur Vereinbarung von metasprachlichen Variablen.

| oder (im Sinne von AUT); steht zwischen alternativen Ausdrücken.

⟨ ⟩ Variablenklammern; der durch diese Klammern begrenzte Ausdruck ist eine metasprachliche Variable. (In einigen Fällen sind Variable nicht streng definiert; ihre Bedeutung ergibt sich dann aus dem in ⟨ ⟩ eingefaßten Text).

[ ] Optionalklammern; der in diesen Klammern eingeschlossene Ausdruck darf weggelassen werden.

{ } Alternativklammern; die alternativen Ausdrücke stehen untereinander

⦃ ⦄ Listenklammern; der in diesen Klammern eingeschlossene Ausdruck steht für ein oder mehrere durch Kommata getrennte Ausdrücke.

⟨ ⟩^{n}
[ ]^n
⦃ ⦄^{n} Ausdrücke, die durch nebenstehende Klammern begrenzt sind, können mit einer Häufigkeitsangabe versehen sein. Diese gibt die mögliche Anzahl von Wiederholungen einer metasprachlichen Variablen, bzw. von Ausdrücken innerhalb von Optional- bzw. Listenklammern an. Als Wiederholungsangaben können Zahlen von 1 bis ∞ oder der Buchstabe n verwendet werden. Dieser Buchstabe wird dann verwendet, wenn keine genauere Angabe möglich ist, da die Anzahl der möglichen Wiederholungen davon abhängt, in welchem Zusammenhang der Ausdruck verwendet wird. Nähere Angaben finden sich meistens im Text, andernfalls ist die Anzahl der möglichen Wiederholungen so groß, daß die Einschränkung praktisch ohne Bedeutung ist.
Die Endsymbole der Vereinbarungen sind die Elemente des Zeichenvorrats.

2. Zeichenvorrat der TAS-Sprache

⟨Ziffer⟩	::=	0|1|2|3|4|5|6|7|8|9
⟨Buchstabe⟩	::=	A|B|C|D|E|F|G|H|I|J|K|L|M|N|Ø|P|Q|R|S|T|U|V|W|X|Y|Z
⟨TAS-Buchstabe⟩	::=	⟨Buchstabe⟩ |*|&
⟨Sonderzeichen⟩	::=	+|-|=|.|,|'|/|(|)|␣|:|;

␣ Zwischenraum (Space, Blank, Leertaste)

Ø Buchstabe Ø, zum Unterschied von der Ziffer 0 (nicht durchgestrichen)

Die angegebenen Zeichen sind in allen für den TR 440 vorgesehenen Eingabecodes enthalten. Beliebige weitere Zeichen dürfen nur in bestimmten Ausdrücken (Oktadenfolgen, Kommentare) vorkommen.

Ignorezeichen (NUL-Oktaden) werden vom Assembler überlesen.
Der Zwischenraum (Blank, Leertaste, Space):
Außer in Oktadenfolgen und Kommentaren ist eine Folge von Blanks einem einzelnen Blank gleichwertig; in den Vereinbarungen (für Metavariable) wird für ein oder mehrere Blanks das Symbol ␣ (Trennstelle) verwendet, wenn an der betreffenden Stelle mindestens ein Zwischenraum erforderlich ist (siehe z.B. B.7).
Außerdem können in eine TAS-Quelle bedeutungslose Blanks eingestreut werden, z.B. vor oder nach TAS-Bausteinen (siehe B.4) und vor oder nach den Sonderzeichen + | - | . |, |/ |(|) ; Sonderzeichenkombinationen wie --|"|+ (dürfen nicht getrennt werden.

3. Formatfreiheit

TAS ist formatfrei; es gibt keine gekennzeichneten Positionen oder Teilbereiche der Eingabezeichen für gewisse Sprachelemente.
Das Zeilenende (Kartenende, Dateisatzende) hat keine Bedeutung und wird übergangen.
Eine TAS-Quelle ist also ein durch Aneinanderfügen der Eingabezeilen entstandener Zeichenstring, der durch den ENDE-Befehl (Kap. C) beendet oder durch Materialende abgebrochen wird.

4. Elemente der TAS-Sprache

TAS-Quellenprogramme sind aus einzelnen Bausteinen, den sogenannten Informationseinheiten, aufgebaut. Diese stellen entweder TR 440-Befehle, TR 440-Konstanten (im folgenden, falls Verwechslungen ausgeschlossen sind, auch kurz als Befehle und Konstanten bezeichnet) oder sogenannte Pseudobefehle in der TAS-Quellensprache dar. Bei Übersetzung des Quellenprogramms werden die Informationseinheiten der Reihe nach vom Assembler interpretiert. Befehle und Konstanten werden in die interne Darstellungsform übersetzt. Die Pseudobefehle sind Anweisungen an den Assembler. Sie werden während der Übersetzung ausgeführt und beeinflussen den Übersetzungsvorgang. Die Informationseinheiten bestehen, abgesehen vom Kommentar, der nur im Protokoll erscheint und nicht übersetzt wird, aus einzelnen Ausdrücken, die durch bestimmte Trennzeichen, z.B. Komma, Vorzeichen, Trennstelle usw. voneinander getrennt sind.
Zwei Arten von Ausdrücken können unterschieden werden: Namen und selbstdefinierende Ausdrücke.

4.1. Namen

⟨Name⟩ ::= ⟨TAS-Buchstabe⟩ [⟨TAS-Buchstabe⟩|⟨Ziffer⟩]ⁿ

Durch Namen werden verschiedene TAS-Objekte, z.B. Adressen oder Adressenzonen, identifiziert. Die Zeichen * und & sind für speziellen Gebrauch vorbehalten (siehe B.4.2) und sollen sonst nicht verwendet werden.
Zonennamen, Montageobjektnamen und Kontaktnamen dürfen bis zu 12 Zeichen, Gebietsnamen bis zu 6 Zeichen lang sein. Alle übrigen Namen dürfen bis zu 31 Zeichen lang sein.

4.2. Sondernamen

Sondernamen unterscheiden sich von den übrigen Namen dadurch, daß in ihnen mindestens eines der Zeichen * oder & vorkommt. Die Sondernamen wurden eingeführt, um zu verhindern, daß Namen von allgemeiner Bedeutung zufällig mit anderen Namen übereinstimmen. Die Namen der Operatoren und Montageobjekte des Programmiersystems und der Dateien der öffentlichen Datenbasis sind (fast alle) Sondernamen.
Vermeidet man Sondernamen, insbesondere als MO- oder Dateinamen, so schützt man sich vor unerwünschten Effekten.

4.3. Selbstdefinierende Ausdrücke

⟨ selbstdefinierender Ausdruck⟩	::=	{ ⟨Zahl⟩ ⟨Tetradenfolge⟩ ⟨Oktadenfolge⟩ ⟨Textfolge⟩ }
⟨ Zahl ⟩		siehe Kapitel B.4.4
⟨ Tetradenfolge ⟩		siehe Kapitel B.4.5
⟨ Oktadenfolge ⟩		siehe Kapitel B.4.6
⟨ Textfolge ⟩		siehe Kapitel B.4.7

Selbstdefinierende Ausdrücke repräsentieren bestimmte Werte, die unmittelbar aus dem Ausdruck ersichtlich sind.

4.4. Zahlen

⟨Zahl⟩	::=	⟨Ziffer⟩^{n}

Zahlen werden wie üblich als Folgen von Ziffern geschrieben.

4.5. Tetradenfolge

⟨Tetradenfolge⟩	::=	'⟨Tetrade⟩^{n} '
⟨Tetrade⟩	::=	⟨Sedezimalziffer⟩
⟨Sedezimalziffer⟩	::=	0| 1| 2| 3| 4| 5| 6| 7| 8| 9| A| B| C| D| E| F

Tetraden sind Folgen von 4 Bits und können daher Werte zwischen 0 und 15 annehmen. Sie werden durch die sogenannten Sedezimalziffern dargestellt. Unter Sedezimalziffern versteht man die Ziffern von 0 bis 9 und die Buchstaben A bis F. Dabei dienen die Buchstaben zur Darstellung der Tetraden mit den Werten von 10 bis 15. Tetradenfolgen sind beliebige Folgen von Tetraden, die durch einzelne Apostrophe eingeschlossen sind. Sie werden vom Assembler in die entsprechenden (durch die einzelnen Tetraden angegebenen) Bitfolgen übersetzt. Tetradenfolgen dürfen (um die Übersichtlichkeit zu erhöhen) durch eingeschobene Blanks gegliedert sein. Diese Leerzeichen werden bei der Übersetzung nicht berücksichtigt.

Beispiel

```
'1A'
'FFFFFF    00F000'
```

4.6. Oktadenfolge

```
⟨Oktadenfolge⟩      ::= "{⟨Zeichen⟩|⟨*-Ersatzzeichen⟩}ⁿ "
⟨*-Ersatzzeichen⟩ ::= *⟨Ziffer⟩³
```

Oktadenfolgen bestehen aus einer in doppelte Apostrophe eingeschlossenen Folge von Zeichen und *-Ersatzzeichen. Jedes Zeichen dieser Folge repräsentiert diejenige Oktade, die zur internen Darstellung des Zeichens verwendet wird. Oktaden sind Folgen von 8 Bits und dienen zur internen Darstellung der Ein- und Ausgabezeichen; die Zuordnung der einzelnen Zeichen zu den einzelnen Oktaden ist im TR 440-Zentralcode festgelegt. Ein *-Ersatzzeichen beschreibt eine Oktade durch ihren Zahlenwert. Dieser Wert wird als dreistellige Dezimalzahl unmittelbar hinter dem * angegeben und muß kleiner als 256 sein. Da zur Abgrenzung der Oktadenfolgen zwei Apostrophe verwendet werden, darf eine Oktadenfolge nur einzelne zusätzliche Apostrophe enthalten. Die Zeichenfolge +(ist in Oktadenfolgen nur als Beginn einer Makrovariablen zulässig. Bei der Bildung von *-Ersatzzeichen dürfen Makrovariable nicht verwendet werden.

Beispiele

```
''A''
''A+B=20''
''A*000B''      :IGNORE-ZEICHEN EINGEFUEGT
''*1201972''    :ERGIBT DIE FOLGE DER 5 OKTADEN *,1,9,7 UND 2
''*1972''       :ERGIBT DIE FOLGE DER 2 OKTADEN F UND 2
''*19--''       :ERGIBT DIE FOLGE DER 5 OKTADEN *,1,9,- UND -
```

4.7. Textfolge

```
⟨Textfolge⟩      ::=  '( { ⟨Baustein⟩ }^n )'

⟨Baustein⟩       ::=  { ⟨Oktadenfolge⟩
                      | ⟨Oktadenname⟩
                      | ⟨Tetradenfolge⟩
                      | ⟨Dezimalzahl⟩ }

⟨Oktadenname⟩    ::=  ⟨Name⟩
```

Die Textfolge ist eine Erweiterung der Oktadenfolge. Sie kann außer Oktadenfolgen noch andere Darstellungsfolgen für einzelne Oktaden enthalten, insbesondere für Oktaden, denen keine Eingabezeichen zugeordnet sind.

Die einzelne Oktade kann entweder durch eine gleichwertige Dezimalzahl oder eine gleichwertige Tetradenfolge dargestellt werden. Darüber hinaus können für einzelne Oktaden mit Hilfe des TEXT-Befehls (siehe C. 9. 1) Namen vereinbart werden und zur symbolischen Darstellung dieser Oktaden verwendet werden. Die Vereinbarung muß der Verwendung vorangehen. Da jede Oktade eine Folge von 8 Bits ist, muß der Wert jeglicher Oktadendarstellung zwischen 0 und 255 liegen.
Außerhalb von Textfolgen sind Oktadennamen nicht verwendbar.

Textfolgen werden durch die Zeichenfolge '(eingeleitet und durch die Zeichenfolge)' beendet. Die einzelnen Bausteine der Textfolge (Oktadenfolgen und die eben beschriebenen Ersatzausdrücke für Oktaden) werden innerhalb der Textfolgen durch Kommata getrennt.

Beispiel

```
NF=  TEXT 23,
SP=  TEXT 175,
     '(NF,''NEUE'',SP,'DC',''BERSCHRIFT'')',
```

5. Informationseinheiten

Die Informationseinheiten sind die selbständigen Bauteile, aus denen TAS-Quellenprogramme zusammengesetzt sind. Sie dienen zur Darstellung von Befehlen, Konstanten und Pseudobefehlen und werden bei der Übersetzung von Quellenprogrammen der Reihe nach interpretiert.

Die Länge der Informationseinheit ist auf maximal 160 Zeichen (einschließlich relevanter Leerzeichen) beschränkt.

5.1. Aufbau der Informationseinheiten

⟨Informationseinheit⟩	::=	[⟨Benennungsteil⟩][⟨Informationsteil⟩ [⟨Spezifikationsteil⟩]] ⟨Abgrenzungsteil⟩
⟨Benennungsteil⟩	::=	⟨Benennung⟩^{n}
⟨Benennung⟩	::=	⟨globale Benennung⟩|⟨lokale Benennung⟩
⟨globale Benennung⟩	::=	⟨Name⟩. =
⟨lokale Benennung⟩	::=	⟨Name⟩=
⟨Informationsteil⟩	::=	{ ⟨Befehl⟩ | ⟨Pseudobefehl⟩ | ⟨Konstante⟩ }
⟨Befehl⟩		siehe Kapitel B.7
⟨Pseudobefehl⟩		siehe Kapitel B.9
⟨Konstante⟩		siehe Kapitel B.10
⟨Spezifikationsteil⟩	::=	/⟨IE-Spezifikation⟩^{n}
⟨IE-Spezifikation⟩	::=	K|V|B|D|O|1|2|3|A|M|N|H|L|R|U|G
⟨Abgrenzungsteil⟩	::=	,|⟨Kommentar⟩|⟨Überschrift⟩
⟨Kommentar⟩		siehe Kapitel B.11
⟨Überschrift⟩		siehe Kapitel B.11

Bei den Informationseinheiten unterscheidet man die Teile: Benennungsteil, Spezifikationsteil und Abgrenzungsteil. Bestimmte Teile können oder müssen je nach Art der Informationseinheit fehlen oder angegeben werden.

5.2. Benennungsteil

Der Benennungsteil besteht aus einem oder mehreren (eventuell mit einem Punkt versehenen) Namen, wobei jeder Name durch Gleichheitszeichen abgeschlossen ist. Diese Namen werden je nach Art der Informationseinheiten verschieden interpretiert.

Die Namen, die im Benennungsteil von Befehlen und Konstanten auftreten, sind den Ablageadressen der Befehle bzw. Konstanten zugeordnet und können daher an beliebigen Stellen des Programms anstelle der Ablageadressen verwendet werden. Je nachdem ob diese Namen im Benennungsteil mit einem Punkt versehen sind oder nicht, haben sie einen globalen oder lokalen Geltungsbereich (siehe C. 3. 1). Der Benennungsteil von Befehlen und Konstanten darf fehlen; er darf höchstens 16 Namen enthalten (bei langen Namen weniger).
Bei den meisten Pseudobefehlen ist im Gegensatz zu Befehlen und Konstanten vorgeschrieben, ob sie einen Benennungsteil haben müssen oder keinen haben dürfen. Bei bestimmten Pseudobefehlen muß der Benennungsteil genau einen Namen enthalten.
Besteht eine Informationseinheit nur aus Benennungs- und Abgrenzungsteil, so überträgt der Assembler den Benennungsteil auf die folgende Einheit.

5.3. Informationsteil

Der Informationsteil bestimmt die Art der Informationseinheit. Er stellt entweder eine Konstante, einen Befehl oder einen Pseudobefehl dar.

5.4. Spezifikationsteil

Der Spezifikationsteil enthält Spezifikationen, die im allgemeinen spezielle, von der Norm abweichende Eigenschaften der Informationseinheit angeben.

5.5. Abgrenzungsteil

Jede Informationseinheit muß durch einen Abgrenzungsteil abgeschlossen sein. Der Abgrenzungsteil ist entweder ein Komma, ein Kommentar oder eine Überschrift. Mittels des Kommentars kann der Programmierer das Quellenprogramm kommentieren. Für die Übersetzung des Quellenprogramms ist der Kommentar jedoch ohne Bedeutung. Eine Informationseinheit kann auch nur aus Kommentar bestehen und dient dann nur zur Beschreibung des Quellenprogramms. Überschriften sind spezielle Kommentare.

6. IE-Spezifikationen

Von der Norm abweichende Eigenschaften einzelner Konstanten und Befehle können mit Hilfe der folgenden Spezifikationen angegeben werden.
Die Spezifikationen können in beliebiger Reihenfolge im Spezifikationsteil der Informationseinheit angeordnet sein.
Bei der Beschreibung der Befehle (Kapitel B. 7), der Konstanten (Kapitel B. 10) und der Pseudobefehle (Kapitel B. 9) ist angegeben, welche der Spezifikationen erlaubt sind (dabei werden auch Beispiele angegeben).

6. 1. Ablagespezifikationen (K, V, B, D)

Diese Spezifikationen sind alternativ. Konstanten und Befehle, die mit den Spezifikationen K, V, B oder D versehen sind, werden in bestimmte Bereiche des Kernspeichers, den sogenannten Konstanten-, Variablen-, Befehls- und Datenbereichen abgelegt. Näheres siehe Kapitel C. 5.

6. 2. Typenkennungsspezifikationen (0, 1, 2, 3)

Jeweils eine der Spezifikationen 0, 1, 2 und 3 darf angegeben werden. Sie bewirkt, daß das Ganzwort, in dem die Konstante bzw. der Befehl abgelegt wird, die angegebene Typenkennung erhält.

Die Typenkennungsspezifikation wird bei Halbworten nur dann wirksam, wenn die Ablageadresse gerade ist. Bei ungerader Ablageadresse richtet sich die Ablagekennung auch bei expliziter Typenkennungsangabe nach dem linken Halbwort.

6. 3. Gerade- und Ungeradespezifikationen (G, U)

Diese Spezifikationen sind alternativ. Die Spezifikation G bzw. U bewirkt, daß Befehle, Halbwort- und Adressenkonstanten auf gerade bzw. ungerade Adressen abgelegt werden. Bei Bedarf wird vom Assembler ein Halbwort eingeschoben um die gewünschte Ablage zu ermöglichen.

6.4. Adressenkonstantenspezifikation (A)

Jede Adressenkonstante (siehe B.10) muß mit der Spezifikation A (die nur zur Kennzeichnung von Adressenkonstanten verwendet werden darf) versehen sein.

6.5. Markierungsspezifikation (M)

Die Spezifikation M dient zur Markierung von Festpunkt- und Adressenkonstanten sowie Gleitpunktkonstanten einfacher und doppelter Genauigkeit. Bei Festpunkt- und Gleitpunktkonstanten wird das erste Bit des Ganzwortes bzw. des Doppelwortes besetzt. Bei Adressenkonstanten wird das erste Bit des Halbwortes besetzt.

6.6. Zweitmarkierungsspezifikation (N)

Die Spezifikation N ist nur bei Adressenkonstanten zulässig. Sie bewirkt, daß das 2. Bit der Adressenkonstante besetzt wird.

6.7. Rechts- und Linksspezifikation (R, L)

Diese Spezifikationen sind alternativ. Die Spezifikationen R und L sind nur bei Tetraden-, Oktaden- oder Textkonstanten erlaubt. Sie bewirken, daß die Tetraden-, Oktaden- oder Textfolgen, die diese Konstanten darstellen, rechts- bzw. linksbündig in Halb- oder Ganzworten abgelegt werden. Bei Bedarf wird der Rest mit Nullen (Tetrade 0) bzw. Ignorezeichen (Oktadenwert 0) aufgefüllt.

6.8. Halbwortspezifikation (H)

Die Spezifikation H ist nur bei Festpunkt-, Oktaden-, Text- und Tetradenkonstanten erlaubt, die intern durch Halbworte dargestellt werden können. Konstanten , die mit dieser Spezifikation versehen sind (sogenannte Halbwortkonstanten), werden in einem Halbwort abgelegt.

7. Befehle

```
⟨Befehl⟩           ::=  ⟨Operationsteil⟩ ␣ ⟨Adressenteil⟩
⟨Operationsteil⟩   ::=  ⟨Befehlscode⟩
⟨Befehlscode⟩      ::=  ⟨Externcode⟩ | ⟨TAS-Interncode⟩
⟨Externcode⟩       ::=  ⟨Name der TR 440-Befehlsliste⟩
⟨TAS-Interncode⟩   ::=  I'⟨Tetrade⟩² '(⟨Externcode⟩)
⟨Adressenteil⟩     ::=  { ⟨Ganzadresse⟩
                        | ⟨Literal⟩
                        | ⟨Rechtsadresse⟩
                        | ⟨Linksadresse⟩ ␣ ⟨Rechtsadresse⟩ }
```

In der Quellensprache dargestellte Befehle bestehen aus Operations- und Adressenteil. Für den Operationsteil wird ein Befehlscode angegeben. Befehle werden vom Übersetzer in die interne Darstellungsform übersetzt. Der implizite Ablagebereich ist B (schreibgeschützt), die implizite Typenkennung ist 2. Liegt bei Ausführung des Programms ein Befehl in einem Ganzwort mit TK ≠ 2, so entsteht ein Befehlsalarm.
Abweichungen vom Normalfall lassen sich außerdem durch den STARR-Befehl (siehe C. 6. 2) oder durch den ABLAGE-Befehl (siehe C. 6. 4) angeben.

7. 1. Befehlscode

Als Befehlscode kann der Externcode oder der TAS-Interncode benutzt werden. Die Kenntnis der TR 440-Befehle wird hier vorausgesetzt. Im "Befehls-Lexikon" und in Kapitel J ist ihre Wirkung beschrieben. Der TAS-Interncode wird nur in seltenen Fällen, z. B. beim Aufbau spezieller Versorgungsblöcke, verwendet. Mit Hilfe des TAS-Interncodes kann die Klassifizierung des Adressenteils unabhängig von der Angabe des Interncodes vorgenommen werden. Der TAS-Interncode enthält daher eine Tetradenfolge, die den Interncode angibt und einen Externcode, der den Adressenteil klassifiziert (und z. B. angibt, ob es sich beim Adressenteil um eine Ganzadresse oder um eine bestimmte Rechtsadresse handelt). Auf Grund beider Angaben übersetzt der Assembler den Befehl so, als stünde als Befehlscode der Externcode und setzt als Codeteil des übersetzten Befehls den angegebenen Interncode ein.

Bei Befehlseinheiten die vom TAS-Übersetzer die Fehlermeldung "Kein Befehl", "Kein Zweitcodebefehl", "Name nicht definiert", "Adreßteil fehlt", "Zuviele Adreßteile" oder "Kein Interncode" erhalten, wird der Interncode '9E' eingesetzt, der im Objektlauf Makroalarm erzeugt.

Im Externcode darf anstelle einer 0 auch O geschrieben werden, hingegen ist 0 anstelle eines O ein Fehler.

Beispiele

```
BA     20,                    Befehl mit Externcode
I'00'  (TXX)  X1  X2,         Befehl mit Interncode
```

7.2. Adressenteile

Die Adressenteile von Befehlen werden vom Assembler je nach der Art des Befehlscodes als Ganzadressen, Rechtsadressen oder Rechts- und Linksadressen interpretiert. Anstelle von Ganzadressen dürfen auch Literale stehen.

Ein Rechtsadreßteil, der nicht bei einem Zweitcodebefehl steht, darf syntaktisch nicht vorzeichenbehaftet sein. Statt eines Minuszeichens ist in diesem Fall vor oder hinter die Dezimalzahl (Rechtsadreßteil) der Buchstabe N zu setzen. Die einzige Möglichkeit für einen vorzeichenbehafteten Rechtsadreßteil, der nicht auf einen Zweitcode folgt, besteht beim Befehl SEGG; z.B. SEGG 3 10N.

Beim Niederschreiben von Befehlen in TAS dürfen Adreßteile nicht weggelassen werden, auch wenn eine Wirkung des Befehls (bei leerem Adreßteil) definiert sein sollte (z.B. bei den Befehlen TRX, IR). Man schreibt dann den Adreßteil in der Form '0' (siehe auch B.7.6).

7.3. Ganzadresse

```
⟨Ganzadresse⟩          ::= ⟨Absolutbezug⟩|⟨Kernspeicherbezug⟩

⟨Absolutbezug⟩         ::= { ⟨einfacher Absolutbezug⟩ [⟨VZ⟩ ⟨einfacher Absolutbezug⟩]ⁿ
                           { ⟨Differenzbezug⟩ [+⟨Differenzbezug⟩]ⁿ

⟨Kernspeicherbezug⟩    ::= ⟨direkter Kernspeicherbezug⟩|⟨Relativbezug⟩

⟨einfacher Absolut-
 bezug⟩                ::= { ⟨selbstdefinierender Ausdruck⟩
                           { ⟨Indexname⟩

⟨Differenzbezug⟩       ::= ⟨symbolische Kernspeicheradresse⟩
                           [⟨VZ⟩ ⟨einfacher Absolutbezug⟩]ⁿ
                           - ⟨symbolische Kernspeicheradresse⟩
                           [⟨VZ⟩ ⟨einfacher Absolutbezug⟩]ⁿ

⟨direkter Kern-
 speicherbezug⟩        ::= [⟨Differenzbezug⟩+]ⁿ ⟨symbolische Kernspeicheradresse⟩
                           [⟨VZ⟩ ⟨einfacher Absolutbezug⟩]ⁿ

⟨Relativbezug⟩         ::= [⟨VZ⟩] ⟨Zahl⟩ R

⟨selbstdefinierender
 Ausdruck⟩                 siehe Kapitel B.4.3

⟨Indexname⟩            ::= ⟨Name⟩

⟨symbolische Kern-
 speicheradresse⟩      ::= ⟨Name⟩

⟨VZ⟩                   ::= +|-
```

Ganzadressen kommen im Adressenteil von Befehlen und in Adressenkonstanten vor. Sie repräsentieren im Adressenteil von Befehlen Zahlen zwischen 0 und $2^{16}-1$ und in den anderen beiden Fällen Zahlen zwischen 0 und $2^{22}-1$.

7.4. Absolutbezüge

Absolutbezüge werden bereits durch den Assembler in ihre endgültige Maschinendarstellung übersetzt (siehe dagegen B. 7. 5). Dabei erhalten selbstdefinierende Ausdrücke den ihnen entsprechenden Zahlenwert. Der Zahlenwert von Indexnamen ist die ihnen zugeordnete Indexadresse.

Symbolische Kernspeicheradressen dürfen in Absolutbezügen nur gepaart auftreten (Differenzbezug). Der erste Komponente des Paares muß ein positives oder kein Vorzeichen haben. Die zweite Komponente muß ein negatives Vorzeichen haben und sich auf dieselbe Adressenzone beziehen wie die erste Komponente. Zwischen den Komponenten dürfen höchstens einfache Absolutbezüge stehen. Der Wert der Differenz ist die (von einer Adressentranslation unabhängige) Differenz der zugeordneten Kernspeicheradressen.

Die sonstigen Subtraktionen und die Addition sind in der gewöhnlichen Weise zu verstehen. Dabei muß der Wert der Konstanten und der Zwischenergebnisse bei der Auswertung von links nach rechts im Bereich -2^{48} bis $+2^{48}$ bleiben. Das Endergebnis jedoch muß in dem Bereich für Ganzadressen liegen (siehe Kapitel B. 7. 3).

7.5. Kernspeicherbezüge

Die einem Kernspeicherbezug zugeordnete Zahl hat die Bedeutung einer Kernspeicheradresse. Sie kann daher erst bei der Montage ermittelt werden. Der Assembler ermittelt nur den zonenrelativen Adressenwert. Wenn Kernspeicherbezüge als Adressenteile von Befehlen auftreten, werden solche Bezüge auch auf ihre Zulässigkeit geprüft. Beispielsweise ist ein Bezug auf eine schreibgeschützte Zone in einem Abspeicherbefehl ein Fehler.

Symbolische Kernspeicheradressen sind Namen, die als Linksbenennungen von Befehlen, Konstanten oder den Pseudobefehlen ASP und DSP auftreten.

Ein Relativbezug ist eine ganze Zahl, evtl. mit Vorzeichen, der ein R angefügt ist. Relativbezüge können nur im Adressenteil von Befehlen auftreten. Sie adressieren den Kernspeicher relativ zu der Speicheradresse des Befehls. Relativbezüge dürfen aus syntaktischen Gründen nicht mit anderen Ausdrücken arithmetisch verknüpft werden.

Beispiele

```
        SI    KON3,      }
B=      SK    A-3,       }  Kernspeicherbezug
        MC    T3+2,      }
        B     4,         }
        BAR   '12',      }  Absolutbezug
A=      XBA   B-A+20,    }
        BA    ''+'',     }
        SK    -2R,          Relativbezug
```

7.6. Links- und Rechtsadressen

⟨Linksadresse⟩	::=	⟨Indexbezug⟩|⟨Parameter⟩|⟨Spezifikation⟩|⟨Zweitcode⟩| ⟨Kernspeicherbezug⟩|'⟨Tetrade⟩²'
⟨Rechtsadresse⟩	::=	⟨Indexbezug⟩|⟨Parameter⟩|⟨Spezifikation⟩|'⟨Tetrade⟩²'
⟨Indexbezug⟩	::=	⟨Absolutbezug⟩
⟨Parameter⟩	::=	[⟨VZ⟩] ⟨Absolutbezug⟩
⟨Zweitcode⟩	::=	⟨Externcode⟩
⟨Spezifikation⟩	::=	{ ⟨Buchstabe⟩ / ⟨Ziffer⟩ }ⁿ
⟨VZ⟩	::=	+|-

Links- und Rechtsadressenteile werden vom Assembler (abhängig von der Art des Befehlscodes, der den Adressenteil klassifiziert) entweder als Indexbezüge, Parameter, Spezifikationen, Zweitcodes oder Kernspeicherbezüge interpretiert und in die Internform (eine zweistellige Sedezimalzahl) übersetzt. Der Assembler kontrolliert dabei (gleichfalls abhängig von der Art des Befehlscodes), ob die Parameter im erlaubten Bereich liegen und ob die Zweitcodes, Spezifikationen und Kernspeicherbezüge zugelassen sind.
Kernspeicherbezüge, die als Linksadressen auftreten, werden intern durch die Differenz zwischen Zieladresse (d.i. die Adresse, auf die Bezug genommen wird) und Adresse des bezugnehmenden Befehls dargestellt. Diese Differenz muß zwischen -127 und +127 liegen.

Anstelle eines Kernspeicherbezugs darf auch ein Parameter stehen, der diese Differenz als Absolut-Bezug darstellt.
Jede Links- und Rechtsadresse kann auch in der internen Form als zweistellige Tetradenfolge dargestellt werden. In diesem Fall entfallen bei der Übersetzung alle Prüfungen auf Zulässigkeit.

7.7. Indexbezug

Ein Indexbezug stellt eine Indexadresse dar, also eine Zahl i, die zwischen 0 und 255 liegt, und adressiert die i-te Zelle der Indexzone (siehe **Kapitel** C.8). Diese Zahl wird durch einen Absolutbezug dargestellt.

In Indexbezügen dürfen auch Namen auftreten, die weder Index- noch Kernspeicheradressen benennen. Diese Namen werden implizit der **Reihe** nach den freien Indexadressen zugeordnet (siehe Kapitel C.8).

Beispiele

```
        SH    ZL     3,     }
        SH    QLKA   10,    } Spezifikation, Parameter
        SH    'F2'   10,    }
        E     B      XX,      Zweitcode, Indexbezug
A=      R     AUT    Q,       Zweitcode, Spezifikation
        EMU   B3     12,    }
        MU    '36'   1,     } Zweitcode, Parameter
        ST    A+2    3A,      Kernspeicherbezug, Spezifikation
        RT    AQ,              Spezifikation
```

8. Literale

```
⟨Literal⟩ ::= ([⟨Informationseinheit⟩]ⁿ ⟨Informationseinheit
                                        ohne Abgrenzungsteil⟩)
```

Ein Literal ist eine in Klammern eingeschlossene Folge von Informationseinheiten. Dabei darf der Abgrenzungsteil der letzten im Literal enthaltenen Informationseinheit fehlen, da diese Informationseinheit durch die schließende Klammer abgegrenzt wird.

Literale können bei allen Befehlen, die als Adressenteil eine Ganzadresse haben dürfen, anstelle dieser Ganzadressen stehen. Sie stellen einen Bezug auf die erste im Literal enthaltene Informationseinheit dar. Die Adresse dieser Informationseinheit wird bei der Übersetzung in den Adressenteil des bezugnehmenden Befehls eingesetzt.

Literale ermöglichen also eine zusätzliche Art der Adressierung, die in bestimmten Fällen zur Einsparung von Schreibarbeit und zur Erhöhung der Übersicht angebracht ist. Befehle, die Literale als Adressenteile aufweisen, dürfen keinen Spezifikationsteil enthalten. In Literalen dürfen auch Befehle enthalten sein, die selbst wieder Literale, die sogenannten Literale 2. Ordnung, enthalten. Eine weitere Verschachtelung ist nicht zulässig. Ein Literal, das nicht Bestandteil eines anderen ist, heißt Literal 1. Ordnung.

8.1. Ablage bei Literalen

Informationseinheiten, die in Literalen enthalten sind, werden zwar wie normale Informationseinheiten übersetzt, aber nicht wie normale Informationseinheiten im Adressenraum angeordnet. Die Literale 1. Ordnung werden bereichsweise zusammengefaßt und in einem von den überigen Informationseinheiten getrennt liegenden, zusammenhängenden Teil des Adressenraums abgelegt. Das gleiche gilt für Literale 2. Ordnung.

Abweichungen von dieser Regelung können durch den ABLAGE-Befehl (siehe Kapitel C. 6. 4) erreicht werden.

Der genaue Ablageort für Literale ist unbestimmt. Lediglich die Adressenzone des Ablageorts liegt aufgrund der Literalordnung fest. Innerhalb eines Literals werden Informationseinheiten mit verschiedenen Adressierungbedingungen (siehe Kapitel C. 5. 1) auch in verschiedenen Adressenzonen ablegt.

Fällt bei der Ablage eines Literals der Literalanfang mit einer ungeraden Adresse der entsprechenden Literalzone zusammen, so legt der Assembler den Literalanfang auf die nächste gerade Adresse, wenn die Typenkennung des vorangehenden Halbwortes nicht mit der des Anfangs des abzulegenden Literals übereinstimmt.

8.2. Ganzwortliterale

Enthält ein Literal als einzige Informationseinheit eine unbenannte Konstante, die in ein Ganzwort übersetzt wird und in einem schreibgeschützten Teil des 16-Bit-Adressenraums abgelegt werden soll, so wird diese Konstante nicht abgelegt, wenn bereits eine gleichartige Konstante abgelegt wurde. Als gleichartig werden Konstanten betrachtet, die in identische Bitfolgen übersetzt werden, die gleiche Typenkennung aufweisen und im gleichen Teil des Adressenraums abgelegt werden sollen. Diese Konstanten werden Ganzwortliterale genannt. Ganzwortliterale werden unabhängig davon, ob es sich um Literale 1. oder 2. Ordnung handelt, in einem eigenen, zusammenhängenden Teil des Adressenraums zusammengefaßt.

Beispiele

```
B      (.443),  ----------------  Ganzwortliteral

SG     (ZX   0   X1,
        B    (0.55), ----------  Ganzwortliteral
        C    HSP
        S    K14),

MCF    ZSP,
B      (''BERLIN'',        }  keine
        ''KOELN'',         }  Ganzwortliterale
        ''BONN''),         }  da mehrere Konstanten

SXI    (ZW3=  NL   B3,
              S    ZW4),

BQB    (''KONSTANZ''),     {  kein Ganzwortliteral
                           {  da 2 Ganzworte

C      CA=  0/V), ----------  kein Ganzwortliteral, da die
                              Konstante nicht in einem schreib-
                              geschützten Teil des Adressenraums
                              abgelegt wird
```

9. Pseudobefehle

⟨Pseudobefehl⟩	::=	⟨Pseudobefehlscode⟩ [⟨Pseudobefehlsadressenteil⟩]
⟨Pseudobefehlscode⟩	::=	⟨Name⟩
⟨Pseudobefehlsadressenteil⟩		verschieden je nach Art des Pseudobefehls, siehe spezielle Beschreibungen

Pseudobefehle sind Anweisungen an den TAS-Übersetzer und dienen zur Steuerung des Übersetzungsvorgangs, oder werden in Anweisungen an den Montierer übersetzt (Deklarationen). Im Gegensatz zu den TR 440-Befehlen werden sie vom Assembler nicht in interne TR 440-Befehle übersetzt. Die einzelnen Pseudobefehle werden in verschiedenen Abschnitten dieses Handbuchs dargestellt. Aus der folgenden Liste des Pseudocodes ist ersichtlich wo die einzelnen Pseudobefehle beschrieben sind.

In bezug auf den Bennungsteil und den Spezifikationsteil nehmen die Pseudobefehle eine Sonderstellung ein. Aus der nachstehenden Liste ist zu entnehmen, bei welchen Pseudobefehlen ein Benennungsteil zwingend vorgeschrieben ist und bei welchen er verboten ist. Des weiteren wird aufgezeigt, welche Spezifikationen erlaubt sind.

Benennung	Pseudobefehl	Spezifikation	beschr. i. Kapitel
-	ABLAGE		C. 6. 4
-	AEND		C. 6. 5
-	ALARM		C. 11. 2
W	ASP	K, V, B, D	C. 4. 1
-	CODE		C. 13. 1
ZW	CZONE		C. 5. 3
-	DEF		D. 4. 1
-	DEND		D. 4. 3
-	DRUCK		C. 10. 1
W	DSP	K, V, B, D	C. 4. 2
-	EINGG		C. 7. 4
-	ENDE		C. 3. 2
-	EXTERN		C. 7. 2
-	EXTOPT		C. 7. 3
-	FORM		D. 6. 1
ZW	FZONE		C. 5. 4
-	GEBAN		C. 6. 9
-	GEBIET		C. 5. 6
ZW	GLCH		C. 12. 1
-	HDEF		I. 4. 1
-	INDEX		C. 8. 3
W	KE		H. 4. 1
-	LDEF		I. 4. 2
-	LDEFAB		I. 4. 2

Benennung	Pseudobefehl	Spezifikation	beschr. i. Kapitel
-	LUECKE		C. 5. 7
-	NOTE		C. 10. 4
W	REPL	U, G	C. 14. 1
-	REND		C. 14. 2
W	SEGM		C. 3. 1
-	SONST		D. 7. 2
-	STARR		C. 6. 2
-	START		C. 11. 5
-	STEND		C. 6. 3
-	STRUKT		C. 11. 6
ZW	TEXT		C. 9. 1
-	UNTPR		C. 11. 3
-	UVB		C. 10. 3
-	VEND		D. 7. 3
-	VERS		D. 7. 1
-	VORBES		C. 11. 4
-	WEND		D. 8. 3
ZW	WIED		D. 8. 1
-	XBASIS		C. 11. 1
-	ZEILE		C. 10. 2
-	ZONAN		C. 6. 7
ZW	ZONE		C. 5. 2

ZW = zwingend
- = nicht erlaubt
W = wahlweise

10. Konstanten

⟨Konstante⟩ ::= { ⟨Festpunktkonstante⟩ | ⟨Gleitpunktkonstante⟩ | ⟨Gleitpunktkonstante doppelter Genauigkeit⟩ | ⟨Tetradenkonstante⟩ | ⟨Oktadenkonstante⟩ | ⟨Textkonstante⟩ | ⟨Adressenkonstante⟩ | ⟨Bitfeldkonstante⟩ }

Konstanten stellen im allgemeinen TR 440-Halb- und Ganzwörter in der Quellensprache dar. Oktaden- und Textkonstanten können mehrere Ganzwörter belegen.

Die bei der Ablage von Konstanten betroffenen Ganzwörter erhalten Typenkennungen, die dem Typ der Konstanten entsprechen. Sie werden im Normalfall in einem schreibgeschützten, direkt adressierbaren Teil des Adressenraums (siehe auch Kapitel C.5) unmarkiert abgelegt. Von der Norm abweichende Eigenschaften einzelner Konstanten lassen sich mit Hilfe von Spezifikationen (siehe Kapitel B.6) oder dem ABLAGE-Befehl (siehe C.6.4) angeben. Konstanten verschiedener Typenkennung können nicht Teile desselben Ganzwortes belegen.

10.1. Festpunktkonstante

⟨Festpunktkonstante⟩	::=	⟨echter Bruch⟩|⟨ganze Zahl⟩
⟨echter Bruch⟩	::=	[⟨VZ⟩] [0]ⁿ.⟨Zahl⟩
⟨ganze Zahl⟩	::=	[⟨VZ⟩] ⟨Zahl⟩
⟨VZ⟩	::=	+|-

Festpunktkonstanten werden normalerweise in Ganzworte mit Typenkennung 1 übersetzt (interne Festpunktkonstanten). Abweichungen von der Normalform sind bei Festpunktkonstanten mit Hilfe der Spezifikationen

0, 1, 2 oder 3 sowie
K, V, B oder D sowie
M oder H sowie
G oder U (nur wenn gleichzeitig H)

zu erreichen.

Beispiele

```
K=      -200/3,
        100/H,
K44.=   -0.125/V2,
        B2     (375/3H),   FP-Konstante in Literal
LOG2=   .3010,
```

10.2. Gleitpunktkonstante

```
⟨Gleitpunktkonstante⟩  ::= { ⟨ganze Zahl⟩ | ⟨echter Bruch⟩ | ⟨unechter Bruch⟩ } E [⟨ganze Zahl⟩]

⟨unechter Bruch⟩       ::= [⟨VZ⟩] ⟨Zahl⟩ . ⟨Zahl⟩
```

Gleitpunktkonstanten werden normalerweise in Ganzworte mit Typenkennung 0 (interne Gleitpunktkonstanten) übersetzt. Abweichungen vom Normalfall können mit Hilfe der Spezifikationen

0, 1, 2 oder 3 sowie
K, V, B oder D sowie
M

angegeben werden.

Beispiele

```
G1=   9999E-5/V,
G2=   341.225E,
       -0.5E2,
G4=   3.14159E/M1,
G5=   +475E+13/B,
       18E,
```

10.3. Gleitpunktkonstante doppelter Genauigkeit

```
⟨Gleitpunktkonstante      ::= { ⟨ganze Zahl⟩ | ⟨echter Bruch⟩ | ⟨unechter Bruch⟩ } D [⟨ganze Zahl⟩]
 doppelter Genauigkeit⟩
```

Eine Gleitpunktkonstante doppelter Genauigkeit wird in zwei aufeinanderfolgende TR 440-Wörter übersetzt. Abweichungen vom Normalfall sind bei Gleitpunktkonstanten mit Hilfe der Spezifikation

K, V, B oder D sowie
M

anzugeben.

Das erste Ganzwort erhält die Typenkennung 0, das zweite Typenkennung 1. Eine Markierungsspezifikation wirkt sich nur auf das erste Ganzwort aus.

Beispiele

```
GD1=   987.654321D-2,
GD2=   0.5D+8/M,
GD3=   -75312468532143.15D13/V,
```


10.4. Tetradenkonstante

⟨Tetradenkonstante⟩	::=	⟨Tetradenfolge⟩
⟨Tetradenfolge⟩		siehe Kapitel B.4.5

Eine Tetradenkonstante wird im Normalfall in ein Ganzwort mit TK2 übersetzt. Die Tetradenfolge darf daher höchstens 12 Tetraden (bei Halbwörtern höchstens 6 Tetraden) enthalten. Bei der Übersetzung wird die Tetradenfolge rechtsbündig in das Ganzwort (bzw. Halbwort) eingesetzt, wobei ggf. (wenn die Tetradenfolge weniger als 12 bzw. 6 Tetraden enthält) mit Nullen aufgefüllt wird. Abweichungen vom Normalfall sind bei Tetradenkonstanten mit Hilfe der Spezifikationen

0, 1, 2 oder 3 sowie
K, V, B oder D sowie
L und H sowie
G oder U (nur wenn gleichzeitig H)

angegeben. Bei der Spezifikation L (Tetradenfolge linksbündig einsetzen) wird ggf. nach rechts hin mit Nullen aufgefüllt.

Beispiele

```
MASK1=      '00F00F0FF'/3LV,
            'FFFF',
KONST1=     'AB00'/HL,
            '0EBBE00EBBE0'/3,
```

10.5. Oktadenkonstante

⟨Oktadenkonstante⟩	::= ⟨Oktadenfolge⟩
⟨Oktadenfolge⟩	siehe Kapitel B.4.6

Eine Oktadenkonstante wird im Normalfall in ein Ganzwort oder mehrere Ganzwörter (je nach Länge der Oktadenfolge) mit Typenkennung 3 übersetzt. Die Oktadenfolge wird dabei linksbündig in diesen Ganzworten abgelegt. Bei Bedarf wird das letzte Ganzwort mit Ignorezeichen aufgefüllt. Abweichungen vom Normalfall können mit den Spezifikationen

0, 1, 2 oder 3 sowie
K, V, B oder D sowie
R und H sowie
G und U (nur wenn gleichzeitig H)

angegeben werden.

Eine mit H spezifizierte Oktadenkonstante darf höchstens 3 Zeichen lang sein.

Beispiele

```
DT1=        ''*021  GIB KOMMANDOS'',
DT2=        ''SEITE  1''/V,
CHARLIST=   ''*''/R,''&''/R,''*115''/R,
```

10.6. Textkonstante

⟨Textkonstante⟩	::=	⟨Textfolge⟩
⟨Textfolge⟩		siehe Kapitel B.4.7

Textkonstanten unterscheiden sich von Oktadenkonstanten nur dadurch, daß anstelle von Oktadenfolgen Textfolgen anzugeben sind. Insbesondere gelten bezüglich Spezifikationen und Ablage die Angaben des vorhergehenden Abschnitts.

Beispiele

```
A=      TEXT  192,
TE=     TEXT  37,
A=      '(A,120,30,''8100'',4,TE)'/2,

NR=     TEXT  131,
        '(NR,''ABC'')',

B=      '(1,3,''KX'')'/HV1,
```


10.7. Adressenkonstante

⟨Adressenkonstante⟩	::=	⟨Ganzadresse⟩/A[IE-Spezifikation⟩]ⁿ [1]
⟨Ganzadresse⟩		siehe Kapitel B.7.3 (Einschränkungen s.u.)
⟨IE-Spezifikation⟩		siehe Kapitel B.5.1

Adressenkonstanten werden in Halbwörter übersetzt, die eine 22-Bit-Adresse enthalten. Die 22-Bit-Adresse wird in der Quellensprache durch eine Ganzadresse dargestellt. Im Spezifikationsteil [1] muß die Spezifikation A enthalten sein (um Adressenkonstanten syntaktisch von Befehlen unterscheiden zu können). Adressenkonstanten können zusätzlich durch die Spezifikationen

0, 1, 2 oder 3 sowie
K, V, B oder D sowie
M sowie N sowie
G oder U

gekennzeichnet sein. Adressenkonstanten haben die implizite Typenkennung 2. Mit Hilfe der Spezifikationen M und N können die ersten 2 Bits des Halbwortes besetzt werden.

Bei Absolutbezügen sind Oktaden- und Textfolgen nicht zugelassen; außerdem muß ein nicht-einfacher Absolutbezug mit einem Namen beginnen.

[1]) Um das Wesentliche an der syntaktischen Form der Adressenkonstanten hervorzuheben, wurde in der eingerahmten Syntax der Spezifikationsteil explizit hingeschrieben. Nach B.5.1 wäre formal ein zweiter Spezifikationsteil möglich, was aber ausdrücklich ausgeschlossen werden soll. Die Syntax ist auch insofern nicht korrekt, als die Spezifikation A nicht die erste zu sein braucht.

Beispiele

```
AKONST=   L1+256/A,
AKO.=     M34/AMN,
AZIF=     0/BAUM,
```

10.8. Bitfeldkonstante

⟨Bitfeldkonstante⟩	::=	({⟨Bitfeldangabe⟩})
⟨Bitfeldangabe⟩	::=	⟨Bitfeldinhalt⟩/⟨Bitfeldlänge⟩
⟨Bitfeldinhalt⟩	::=	⟨Absolutbezug⟩
⟨Bitfeldlänge⟩	::=	⟨Zahl⟩
⟨Absolutbezug⟩		siehe Kapitel B.7 und die folgende Erläuterung

Mit Hilfe der Bitfeldkonstanten können beliebige Bitfelder besetzt werden. Eine Bitfeldkonstante besteht aus einer Liste von einzelnen Bitfeldangaben, die jeweils Inhalt und Länge eines Bitfeldes angeben. Eine einzelne Bitfeldkonstante darf maximal ein Ganzwort lang sein, d.h. die Summe der Bitfeldlängen einer Bitfeldkonstanten darf nicht größer als 48 sein.

Die Bitfelder schließen dicht aneinander an; das erste Bitfeld beginnt im ersten Bit (links) im Halb- oder Ganzwort. Der Bitfeldinhalt wird im Bitfeld rechtsbündig abgelegt (überzählige Bits sind nicht gesetzt). Als Bitfeldinhalte sind Absolutbezüge (einschließlich Differenzbezüge) mit Ausnahme der Textfolge zugelassen. Der Wert des Bitfeldinhalts muß ganz im Bitfeld, das durch die zugehörige Bitfeldlängen-Angabe bestimmt ist, unterzubringen sein. Im allgemeinen bekommt die Bitfeldkonstante die Typenkennung 2.

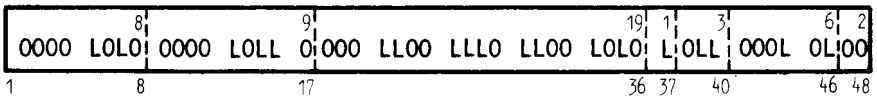
Erlaubte Spezifikationen der Bitfeldkonstanten:

0, 1, 2 oder 3 sowie
K, V, B oder D sowie
H sowie
G oder U (nur wenn gleichzeitig H)

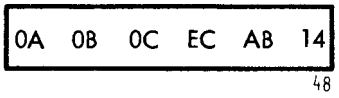
Beispiel

```
(10/8, '16'/9, ''OK''/19, 1/1, 3/3, 5/6),
```

ergibt:



ergibt:



10.9. Oktadenadressierung und OA-Befehl

```
⟨ØA-Befehl⟩ ::= ØA␣ ⟨Parameter⟩␣⟨KSP-Bezug⟩
```

Für die Befehle ZK und TOK sind Oktadenadressen nötig. Diese können mittels Oktadenadreßkonstanten angegeben werden. Eine Oktadenadreßkonstante wird als Befehl mit zwei Adreßteilen geschrieben. Der Parameter mit Werten zwischen 0 und 255 dient zur Angabe der Relativposition des Bytes relativ zum ersten Byte des durch den KSP-Bezug angegebenen Halbwortes.

Als Montagecode erhält man eine HW-Wertbestimmung mit dem Wert (3 x KSP-Bezug) + Parameterwert und dem Translationsschlüssel 4.

Die implizite TK ist 2, der implizite Ablagebereich ist B, d.h. es muß i.a. die Spezifikation K oder V gegeben werden.

Beispiel

```
STR=     ''ABCDEF'',
OAE=     OA  4  STR/K  --OKTADENADRESSE VON E--
```

10.10. EA-Befehlswort

```
⟨VW-Befehl⟩    ::=  VW␣⟨Schlüssel⟩ [␣] ⟨Länge⟩
⟨Schlüssel⟩    ::=  D|F|N
⟨Länge⟩        ::=  ⟨Zahl < 2048⟩
```

Der VW-Befehl liefert eine Halbwortkonstante mit TK2, die als rechte Hälfte eines EA-Befehlswortes Verwendung findet. Die ersten 11 Bit (von links) des Halbwortes enthalten die Abschnittslänge, die im VW-Befehl direkt angegeben wird; die restlichen 13 Bit enthalten bei Angabe des Schlüssels D (Durchstart) das Bitmuster '801', bei F (Fortstart) das Muster '1' und bei N (Neustart) das Muster '1801'.

11. Kommentare und Überschrift

```
⟨Kommentar⟩            ::=  { --⟨Kommentartext Typ1⟩--
                            { :⟨Kommentartext Typ2⟩;

⟨Überschrift⟩          ::=  ---⟨Kommentartext Typ1⟩--

⟨Kommentartext Typ1⟩   ::=  ⟨beliebige Zeichenfolge, die mindestens ein
                             Zeichen enthalten muß, mit keinem Minus-
                             zeichen beginnen oder enden darf und keine
                             zwei aufeinanderfolgende Minuszeichen ent-
                             halten darf⟩

⟨Kommentartext Typ2⟩   ::=  ⟨beliebige Zeichenfolge, die die Zeichen
                             Doppelpunkt oder Semikolon nicht enthalten
                             darf⟩
```

Kommentare dienen nur zur Erläuterung des Quellenprogramms und werden unverändert in das TAS-Übersetzerprotokoll aufgenommen.

Man unterscheidet zwei Arten von Kommentarbegrenzungen. Bei der symmetrischen Kommentarbegrenzung beginnt und endet der Kommentar mit zwei aufeinanderfolgenden Minuszeichen.
Die unsymmetrische Kommentarbegrenzung durch Doppelpunkt und Semikolon hat den Vorteil, daß dem Assembler bei versehentlichem Weglassen eines Grenzzeichens das Aufsetzen erleichtert wird.

Wagenrücklaufzeichen werden im Kommentartext wie im gesamten TAS-Quellentext überlesen. Kommentare dürfen entweder als Abschluß eines Informationsteils oder aber auch als selbständige Informationseinheit auftreten.
Überschriften sind spezielle Kommentare. Sie beginnen mit drei Minuszeichen und werden in die Blattüberschrift des TAS-Übersetzerprotokolls übernommen. Sie bewirken den Beginn einer neuen Seite.

Kommentar bzw. Überschrift sind als Abgrenzungszeichen in Listen (in Pseudobefehl-Adreßteilen) verboten.

Beispiele

```
A=      SH   ZL   10   --KOMMENTAR--
        10, 50, 20,    --SELBSTAENDIGER KOMMENTAR--
B=      SEGM           ---UEBERSCHRIFT--
                       ---SELBSTAENDIGE UEBERSCHRIFT--
        S   L1         :A;
        S   L2         :B;
```

STEUERUNG DES ÜBERSETZUNGSVORGANGES

1.	Einleitung	1
2.	Übersetzung und Montage	1
3.	Gliederung von Quellenprogrammen, Segmente	4
3.1.	SEGM-Befehl	4
3.2.	ENDE-Befehl	6
4.	Vereinbarung von Arbeitsspeichern	6
4.1.	ASP-Befehl	6
4.2.	DSP-Befehl	8
5.	Adressenzonen und Gebiete	8
5.1.	Adressenzonen	9
5.2.	ZONE-Befehl	10
5.3.	CZONE-Befehl	11
5.4.	FZONE-Befehl	11
5.5.	Gebiete	11
5.6.	GEBIET-Befehl	13
5.7.	LUECKE-Befehl	16
6.	Anordnung von Informationseinheiten Adressenzonen und Gebieten	16
6.1.	Verteilung der Informationseinheiten auf die Adressenzonen	16
6.2.	STARR-Befehl	18
6.3.	STEND-Befehl	18
6.4.	ABLAGE-Befehl	19
6.5.	AEND-Befehl	20
6.6.	Anordnung von Adressenzonen in Gebieten	20
6.7.	ZONAN-Befehl	20
6.8.	Anordnung von Gebieten im Adressenraum	21
6.9.	GEBAN-Befehl	22
6.10.	Schematische Übersicht für die Anordnung von Informationseinheiten, Adressenzonen und Gebieten	24

7.	Querbezüge zwischen Quellenprogrammen	25
7.1.	Montageobjektnamen und Kontaktnamen	25
7.2.	EXTERN-Befehl	26
7.3.	EXTOPT-Befehl	27
7.4.	EINGG-Befehl	27
8.	Zuordnung der Indexnamen	28
8.1.	Indexzellen und ihre Verwendung	28
8.2.	Benennung von Indexadressen	29
8.3.	INDEX-Befehl	29
9.	Vereinbarung von Oktadennamen	31
9.1.	TEXT-Befehl	31
10.	Steuerung des TAS-Protokolls	32
10.1.	DRUCK-Befehl	32
10.2.	ZEILE-Befehl	33
10.3.	UVB-Befehl	33
10.4.	NOTE-Befehl	34
11.	Startanweisungen für Operatoren	34
11.1.	XBASIS-Befehl	35
11.2.	ALARM-Befehl	35
11.3.	UNTPR-Befehl	35
11.4.	VORBES-Befehl	36
11.5.	START-Befehl	36
11.6.	STRUKT-Befehl	36
12.	Gleichsetzung	37
12.1.	GLCH-Befehl	37
13.	Codeumsteuerung für Oktadenfolgen	39
13.1.	CODE-Befehl	39
14.	Replikation von Informationseinheiten	39
14.1.	REPL-Befehl	39
14.2.	REND-Befehl	40

STEUERUNG DES ÜBERSETZUNGSVORGANGES

1. Einleitung

In diesem Teil der TAS-Beschreibung wird beschrieben, wie mit Hilfe von Pseudobefehlen Einfluß auf den Übersetzungsvorgang genommen werden kann.

Häufig wird von Informationseinheiten "vor", "nach", "zwischen" anderen Informationseinheiten gesprochen. Diese Angaben beziehen sich immer auf die Reihenfolge der Abarbeitung durch den Assembler. Diese ist nicht in jedem Falle die Reihenfolge in der Quelle. In folgenden Fällen weicht die Abarbeitungsreihenfolge von der Reihenfolge in der Quelle ab.

1. Bei Makrodefinitionen (siehe D. 4)

   Die Informationseinheiten innerhalb einer Makrodefinition werden übergangen.

2. Bei Makroaufrufen (siehe D. 5)

   Hier wird in die Reihenfolge nach dem Makroaufruf der Definitionstext der zugehörigen Makrodefinition eingesetzt.

3. Bei Versionstexten (siehe D. 7)

   Ob ein Versionstext in die Reihenfolge aufgenommen wird oder ob er übergangen wird, entscheidet sich anhand der im VERS-Befehl angegebenen Bedingung.

4. Bei Wiederholungstexten (siehe D. 8)

   Auf den WEND-Befehl folgt die erste Informationseinheit des Wiederholungstextes oder die in der Quelle auf den WEND-Befehl folgende Informationseinheit, je nach Abarbeitungsstand der Werteliste. Der Wiederholungstext wird ganz übergangen, wenn die Werteliste die leere Liste ist.

2. Übersetzung und Montage

Für das Verständnis gewisser Teile der TAS-Sprache ist eine Übersicht darüber notwendig, wie es zu einer maschineninternen Darstellung von Programmen kommt.

Der Programmierer schreibt Quellenprogramme. Quellenprogramme können in verschiedenen Programmiersprachen, z. B. in TAS, verfaßt sein. Für jede Quellensprache gibt es einen Übersetzer. Dieser übersetzt ein Quellenprogramm stets als Ganzes in ein Montageobjekt. Die Darstellungsform des Montageobjekts ist unabhängig von der Quellensprache, d.h. der Sprache, in der das Quellenprogramm verfaßt ist.

Der Montierer ist ein Programm, das mehrere Montageobjekte (im Grenzfall eines) zusammenschließt, in einen lauffähigen Code, den Operatorkörper, übersetzt und für den Abwickler eine Beschreibung dieses Codes, die Operatorkörperbeschreibung, anfertigt.

Eine Gesamtquelle (für einen Operator) ist die Gesamtheit der Quellenprogramme, aus denen ein Operatorkörper entsteht. Die einzelnen Quellenprogramme, aus denen die Gesamtquelle zusammengesetzt ist, brauchen nicht alle in derselben Programmiersprache geschrieben zu sein.

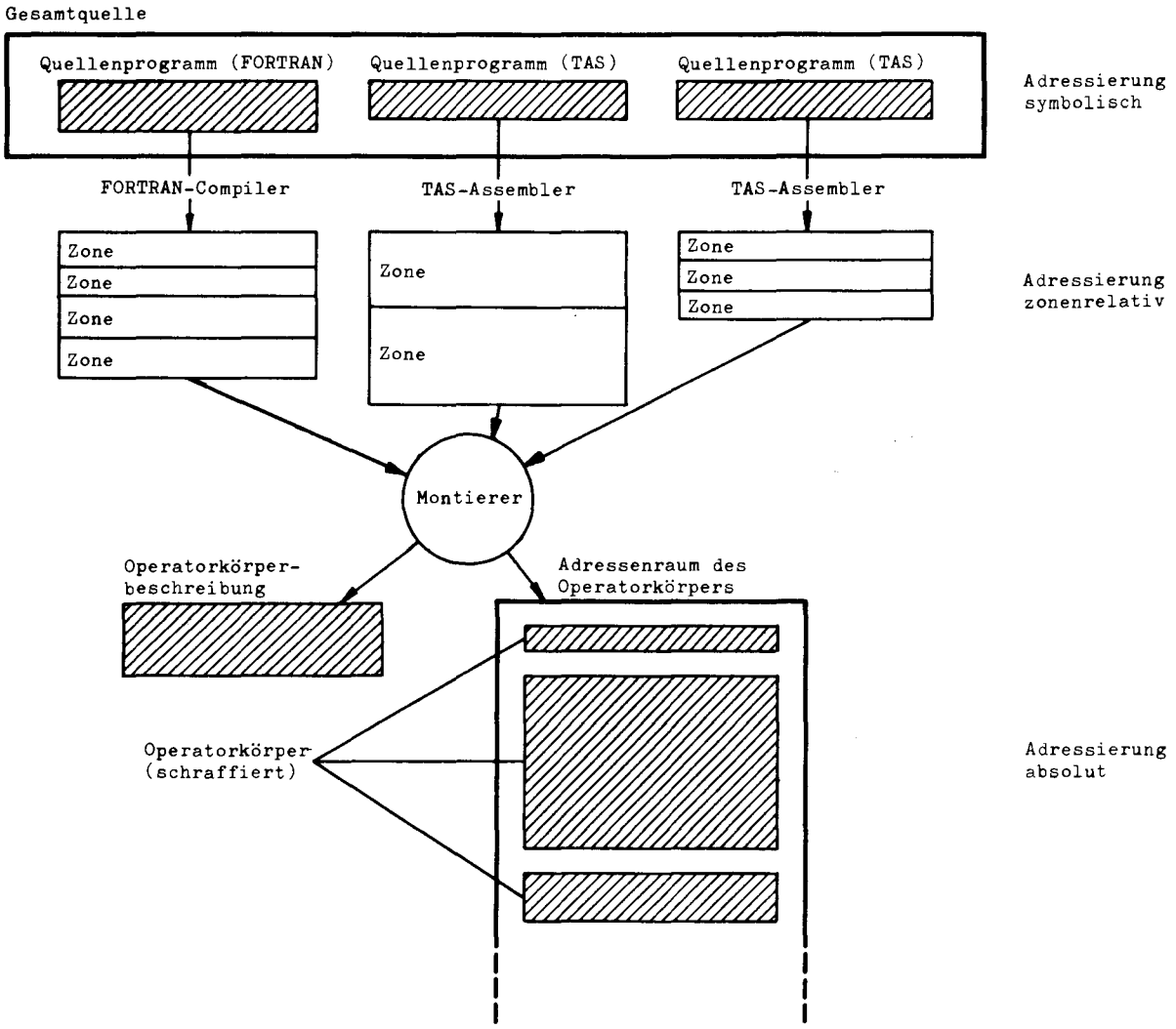
Jeder Operator besitzt einen eigenen Adressenraum. Im Operatorkörper dargestellte Kernspeicheradressen beziehen sich auf diesen Adressenraum. Sie heißen absolute Adressen [1]). Charakteristisch für das Montageobjekt jedoch ist, daß darin vorkommende Kernspeicheradressen noch nicht absolut sind. Kernspeicheradressen im Montagecode sind zonenrelativ. Adressenzonen sind unabhängige Adressenbereiche, die vom Übersetzer vereinbart und dem Montierer im Montagecode mitgeteilt werden. Zu jeder Adressenzone wird vom Übersetzer angegeben, wie lang sie ist und welche Adressierungsbedingungen zu beachten sind (zur Adressierungsbedingung siehe C. 5. 2). Darüber hinaus kann der Übersetzer auch Vorschriften für die relative oder die absolute Lage von Adressenzonen machen (siehe ZONAN C. 6. 7 und GEBAN C. 6. 9).

Der Montierer betrachtet alle Adressenzonen (gleicher Adressierungsbedingung) aus allen Montageobjekten für einen Operatorkörper als gleichwertig und ordnet sie im Adressenraum des Operatorkörpers an. Kernspeicheradressen werden dabei aus der zonenrelativen in die absolute Darstellung translatiert.

---

[1]) Es kommt auf den Gesichtspunkt an, was als "absolut" zu bezeichen ist. Vom Gesichtspunkt des Assemblers und Montierers und damit auch dessen, der Operatoren programmiert, sind Adressen des operatorspezifischen Adressenraums absolut. Für den Abwickler z. B. sind sie dagegen "operatorrelativ".

Im folgenden Diagramm ist der Sachverhalt noch einmal schematisch dargestellt.



3. Gliederung von Quellenprogrammen, Segmente

Ein Quellenprogramm kann aus einem oder mehreren Segmenten bestehen. Ein Segment ist der Geltungsbereich von lokalen Namen.

3.1. SEGM-Befehl

```
⟨SEGM⟩-Befehl   ::= [⟨Segmentname⟩=] SEGM
⟨Segmentname⟩   ::= ⟨Name⟩
```

Der SEGM-Befehl hat zwei grundverschiedene Funktionen, nämlich die Namensgebung für das Quellenprogramm und die Zerlegung des Quellenprogramms in Segmente.

Ist der erste SEGM-Befehl benannt, so wird der erste Name (dieses Befehls) Quellenprogrammname. Ist der erste SEGM-Befehl nicht benannt, so erhält das Quellenprogramm keinen Namen. In der Kopfzeile des Übersetzungsprotokolls bleibt die Position "Programmname" unbesetzt.

Das aus einem Quellenprogramm erzeugte Montageobjekt erhält den Namen des Quellenprogramms. Im Fall eines namenlosen Quellenprogramms wird der Standardname STDHP verwendet. Das vorstehende gilt dann, wenn im UEBERSETZE-Kommando kein anderer Montageobjektname explizit angegeben wurde (siehe Kommandohandbuch,Best. Nr. N31.D0.01).

Durch SEGM-Befehle wird das Quellenprogramm in Segmente zerlegt. Diese sind nur für die Gültigkeit der lokalen Namen (siehe auch B.5) von Bedeutung. Die Vereinbarung eines lokalen Namens gilt immer vom nächsten vorausgehenden SEGM-Befehl bis zum nächsten nachfolgenden SEGM-Befehl bzw. zum Ende des Quellenprogramms. Die Benennung eines SEGM-Befehls ist stets gleichzeitig eine symbolische Adresse für das nächste freie Halbwort in derjenigen Adressenzone, in die ein unmittelbar hinter dem SEGM-Befehl stehender, nicht spezifizierter Befehl abgelegt werden würde.
Namen, die vor dem ersten SEGM-Befehl vereinbart sind, gelten als global.

Beispiele

```
BEISPIEL=    SEGM,

       V=    ASP     100,
             INDEX   8(X1.,X2),

      L1=    XBA     V,
             XC      X1,
             ZX      1       X2,...
             S       L2,
             SEGM,

       K.=   17,
     L2.=    LZL     1       0,
      L2=    EZB     X1,
             AUT     K,
             SNO     L2,...

    ENDE=    SEGM,

             INDEX   (X2,X3),
       V.=   ASP     100,
             ZX      0       X2,

             TXX     X3      X1,

             B       K,...

             S       L2,

             ENDE,
```

Segment Nr.	Name	Lokal Global	gültig in 1	2	3
	BEISPIEL	L	x		
	V	L	x		
1	X1	G	x	x	x
	X2	L	x		
	L1	L	x		
	K	G	x	x	x
2	L2	G	x		x
	L2	L		x	
	ENDE	L			x
3	X2	L			x
	X3	L			x
	V	G		x	x

3.2. ENDE-Befehl

```
⟨ENDE-Befehl⟩  ::=  ENDE
```

Das Ende eines Quellenprogramms wird dem Übersetzer durch den ENDE-Befehl angezeigt. Fehlt dieser Befehl, so wird vom Assembler eine Fehlermeldung gegeben und automatisch ein ENDE-Befehl eingesetzt.

Der ENDE-Befehl wird auch in Makro-, Versionstext- und Wiederholungstext-Definitionen (siehe Kapitel D) erkannt und ausgeführt.

4. Vereinbarung von Arbeitsspeichern

4.1. ASP-Befehl

```
⟨ASP-Befehl⟩          ::= [⟨Benennungsteil⟩] ASP ␣ ⟨Freihalteangabe⟩
                                                  [⟨Spezifikationsteil⟩]

⟨Freihalteangabe⟩     ::= { ⟨Anzahl freizuhaltender Halbworte⟩ }
                          { ⟨Teilseitenzahl⟩⟨Teilseitenangabe⟩ }

⟨ASP-Gleichsetzung⟩   ::= ⟨gleichgesetzer Name, dem ein Wert vom Typ
                           ⟨Freihalteangabe⟩ zugeordnet ist⟩

⟨Teilseitenangabe⟩    ::= { ⟨Ganzseitenangabe⟩      }
                          { ⟨Halbseitenangabe⟩      }
                          { ⟨Viertelseitenangabe⟩   }
                          { ⟨Achtelseitenangabe⟩    }

⟨Teilseitenzahl⟩      ::= ⟨Zahl⟩
⟨Ganzseitenangabe⟩    ::= K
⟨Halbseitenangabe⟩    ::= K2
⟨Viertelseitenangabe⟩ ::= K4
⟨Achtelseitenangabe⟩  ::= K8
⟨Spezifikationsteil⟩      siehe Kapitel B.5.4
```

Der ASP-Befehl dient zur Vereinbarung sogenannter Arbeitsspeicher. Diese Speicher liegen normalerweise in der Zone, zu der auch die Variablen gehören (das sind mit V spezifizierte Größen). Bei der Montage belegt der Montierer diese Bereiche mit dem Vorbesetzungsmuster. Im Benennungsteil des ASP-Befehls können der Anfangsadresse des Arbeitsspeichers lokale und globale Namen zugeordnet werden.

Es wird auf die Gefahr der Typenkollision bei Speichervereinbarungen, die auf ungeraden Adressen beginnen oder auf geraden Adressen enden, hingewiesen (vergl. dazu B. 6. 2, C. 11. 4 und C. 5. 6).

Falls der Arbeitsspeicher auf jeden Fall mit einer geraden oder ungeraden Adresse beginnen soll, ist dies mit den Spezifikationen U oder G anzugeben. Forderungen bezüglich einer von der Norm abweichenden Anordnung des Arbeitsspeichers in einen bestimmten Teil des Adressenraums können mit Hilfe der Ablagespezifikationen K, V, B oder D angegeben werden.
Im Adressenteil des ASP-Befehls wird die Länge des Arbeitsspeichers entweder in Halbworten oder in Teilseiten (Ganz-, Halb-, Viertel- oder Achtelseiten) angegeben. Im letzteren Fall wird der Arbeitsspeicher so im Adressenraum angeordnet, daß sein Beginn entsprechend dem Typ der Teilseitenangabe mit einem Ganz-, Halb-, Viertel- oder Achtelseitenbeginn zusammenfällt.
Die Freihalteangabe kann auch über eine Gleichsetzung angegeben werden; der zugehörige GLCH-Befehl muß aber in der Quelle vorangehen.

Beispiele

```
ARBSP=    ASP    40/G
ZWSP.=    ASP    17,
EASP=     ASP    3K,
TRP=      ASP    0K8/B,
          BA     20,
LNG=      GLCH   5K8,
PUFFER=   ASP    LNG,
```

Man beachte, durch Einschieben des Arbeitsspeichers TRP = ASP OK8/B wird erreicht, daß der Befehl BA 20 auf den Beginn einer neuen Achtelseite gelegt wird.

4.2. DSP-Befehl

⟨DSP-Befehl⟩	::= [⟨Benennungsteil⟩]DSP ␣ ⟨Freihalteangabe⟩ [⟨Spezifikationsteil⟩]
⟨Freihalteangabe⟩	siehe Kapitel C.4.1
⟨Spezifikationsteil⟩	siehe Kapitel B.5.4

Der DSP-Befehl dient zur Vereinbarung von sogenannten Datenspeichern. Diese unterscheiden sich von dem im vorangehenden Abschnitt erklärten Arbeitsspeichern nur darin, daß sie im Normalfall in einem speziellen, nur mit 22-Bit-Adressen ansprechbaren Teil des Adressenraums, angeordnet werden. Für den DSP-Befehl gelten daher sinngemäß die im vorangehenden Abschnitt angegebenen Konventionen des ASP-Befehls.

Beispiele

```
MAT3D.=   DSP    17/U,
DEPOT=    DSP    9K2,
LNG=      GLCH   20,
LISTE=    DSP    LNG/G,
```

5. Adressenzonen und Gebiete

Wegen der unterschiedlichen Adressierungsbedingungen für die einzelnen Informationseinheiten (Schreibschutz, Einhaltung von Großseitengrenzen) können diese nicht einfach hintereinander in der Reihenfolge ihrer Bearbeitung durch den Assembler abgelegt werden. Es wird daher eine Umordnung vorgenommen , die durch eine Sortierung durch den Assembler und ein Zusammenfügen durch den Montierer zustande kommt.

Für die Durchführung und die Beschreibung dieser Umordnung sind die Begriffe der Adressenzone und des Gebiets besonders wichtig.

In diesem Abschnitt wird nur die Vereinbarung von Adressenzonen und Gebieten behandelt. Die Umordnung wird im Kapitel C. 6 beschrieben.

5.1. Adressenzonen

Adressenzonen sind vom Assembler eingeführte, mit 0 beginnende, zusammenhängende Adressenräume. Diese Adressenräume werden vom Montierer durch Addition einer Translationsgröße in den Adressenraum des Operators, für den die Montage erfolgt, abgebildet. Nach der Montage sind die Adressenzonen vergessen.

Der Assembler richtet eine Adressenzone erst bei Bedarf ein oder wenn er durch einen der Pseudobefehle ZONE (C. 5. 2), CZONE (C. 5. 3) oder FZONE (C. 5. 4) dazu aufgefordert wird. Im ersten Fall sind die Adressenzonen "implizit" im letzteren Fall "explizit" definiert.

Es gibt maximal 13 implizit definierte Adressenzonen. Im Gegensatz zu den explizit definierten Adressenzonen haben sie keine Namen und werden nur für den Beschreibungszweck mit K0, K1, K2, KG, V0, V1, V2, B0, B1, B2, D0, D1, D2 bezeichnet.

Jeweils der erste Buchstabe bedeutet eine Adressierungsbedingung, die vom Montierer bei der Anordnung im Adressenraum beachtet wird:

- K: schreibgeschützt, mit 16-Bit-Adressen adressierbar, (Konstanten)
- V: nicht schreibgeschützt, mit 16-Bit-Adressen adressierbar, (Variable)
- B: schreibgeschützt, alle B-Adressenzonen kommen zusammen in eine Großseite (Befehle)
- D: nicht schreibgeschützt, mit 22-Bit-Adressen adressierbar, (Daten)

Die explizit definierten Adressenzonen werden im definierenden Pseudobefehl benannt. Der Name wird keiner Kernspeicherzelle zugeordnet, darf also nicht als KSP-Bezug benutzt werden.

Explizite Zonen sind nur dann sinnvoll, wenn ihnen auch Informationseinheiten durch den ABLAGE-Befehl (siehe C. 6. 4) zugeordnet werden.

Die Anzahl der (explizit und implizit definierten) Adressenzonen ist pro Monatgeobjekt auf 31 begrenzt.

Die Größe jeder Adressenzone steht erst am Schluß des Assemblerlaufs fest und wird durch die Gesamtheit der ihr durch den Assembler zugewiesenen Informationseinheiten bestimmt.

5.2. ZONE-Befehl

```
⟨ZØNE-Befehl⟩              ::= ⟨Zonenname⟩ = ZØNE ␣ ⟨Adressierungsbedingung⟩
⟨Zonenname⟩                ::= ⟨Name⟩
                               ⎧ K|V  ⎫
                               ⎪ B|BV ⎪
⟨Adressierungsbedingung⟩    ::= ⎨ D|DK ⎬
                               ⎪ F|FV ⎪
                               ⎩ I    ⎭
```

Der ZONE-Befehl vereinbart explizit eine Adressenzone und benennt sie. Die Bedeutung der Adressierungsbedingung entspricht für K, V, B, D genau der Bedeutung des ersten Buchstabens der Bezeichnung für implizit definierte Adressenzonen (C. 5. 1).

Darüber hinaus bedeutet:

BV: nicht schreibgeschützt, in derselben Großseite, in der auch die implizit definierten B-Adressenzonen liegen, (Befehle, variabel)

F: schreibgeschützt, innerhalb einer Großseite, (freie Befehlszone)

FV: nicht schreibgeschützt, innerhalb einer Großseite, (freie Befehlszone, variabel)

DK: schreibgeschützt, mit 22-Bit-Adressen adressierbar, (Daten, konstant)

I: Indexzone (nicht schreibgeschützt, mit 16-Bit-Adressen adressierbar). Der Anfang der Indexzone wird mindestens 256 Adressenstellen vor dem Ende des Gebiets abgelegt, in dem die Indexzone angeordnet wird.

Wird die Adressierungsbedingung bei der Anordnung der Zone im Adressenraum (siehe ZONAN-Befehl C. 6. 7 und Anordnung von Gebieten im Adressenraum C. 6. 8) verletzt, so reagiert der Montierer mit einer Warnung, falls nur der Schreibschutz betroffen ist, sonst mit Abbruch des Montagevorgangs.

5.3. CZONE-Befehl


```
⟨CZØNE-Befehl⟩  ::=  ⟨Zonenname⟩ = CZØNE ␣ ⟨Adressierungsbedingung⟩
```

Der CZONE-Befehl vereinbart eine sogenannte Commonzone. Diese unterscheidet sich von der durch ZONE vereinbarten Adressenzone nur dadurch, daß bei der Montage mehrerer übersetzter Quellenprogramme gleichnamige Commonzonen als identisch angesehen und daher nur einmal im Adressenraum eingeordnet werden. Der Umfang des zugewiesenen Adressenraums wird nach der längsten aller gleichnamigen Commonzonen bemessen.

Die Adressierungsbedingung entspricht der beim ZONE-Befehl.

5.4. FZONE-Befehl

```
⟨FZØNE-Befehl⟩  ::=  ⟨Zonenname⟩ = FZØNE ␣ ⟨Adressierungsbedingung⟩
```

Ein FZONE-Befehl vereinbart eine sogenannte Freihaltezone. Diese beansprucht zwar einen Teil des Adressenraums, aber zu Beginn des Operatorlaufs noch keinen physikalischen Speicher. Informationseinheiten, die zu einer Freihaltezone gehören, werden daher nicht in die interne Form übersetzt, sondern dienen nur zur Festlegung der Länge und zur Vereinbarung von Namen für Adressen der Freihaltezone. Die Zugehörigkeit der Informationseinheiten zu Adressenzonen ist in Kapitel C. 6 beschrieben.

Die Adressierungsbedingung entspricht der beim ZONE-Befehl.

5.5. Gebiete

Der Montierer verbindet die zu einem Operator gehörigen Adressenzonen zu einem Operatorkörper, der zusammen mit seiner ebenfalls vom Montierer angelegten Beschreibung eine vom Abwickler startbare Einheit darstellt.

Der Operatorkörper kann jedoch vor dem System im allgemeinen nicht als ungegliedertes Gebilde auftreten. Die Eigentümlichkeiten der Hardware und des Systems bedingen, daß die Adressenzonen mit ihren Adressierungsbedingungen zu einzelnen Verwaltungsobjekten des Systems, den "Gebieten", gruppiert werden müssen. Gebiete tragen (in diesem Zusammenhang) stets eine Adressierung, die ein Vielfaches einer Seite überstreicht, d.h. einem adressierten Gebiet ist ein Ausschnitt des Adressenraums von $m \cdot 2^{11}$ bis $n \cdot 2^{11}-1$ ($m \leq n$; für $m = n$ ist das Gebiet leer) zugeordnet.

Schreibschutz kann nur für ein ganzes Gebiet bewirkt werden. Das gleiche gilt für einige weitere Eigenschaften, die den Parametern des GEBIET-Befehls zu entnehmen sind (siehe Kapitel C. 5. 6).

Für die implizit vereinbarten Adressenzonen und die explizit vereinbarten Adressenzonen, über deren Anordnung nicht explizit verfügt wird (ZONAN-Befehl, C. 6. 7), erstellt der Montierer Gebiete. Diese heißen "implizit definiert". Über die Anzahl und die Lage der implizit definierten Gebiete entscheidet der Montierer aufgrund gewisser Optimierungsstrategien.

5.6. GEBIET-Befehl

```
⟨GEBIET-Befehl⟩            ::= GEBIET [ ( { ⟨Gebietsparameter⟩ }^n ) ]

                                  [ØNAME=]  ⟨operatorspezifischer Gebietsname⟩
                                  [PNAME=]  ⟨prozeßspezifischer Gebietsname⟩
                                  [LD=]     ⟨Lebensdauer⟩
                                  [ADR=]    ⟨Gebiets-Adressierungsbedingung⟩
⟨Gebietsparameter⟩         ::=    [VK1=]    ⟨Verarbeitungsklasse⟩
                                  [VK2=]    ⟨Verarbeitungsklasse⟩
                                  [LK=]     ⟨Lagerklasse⟩
                                  [VB=]     ⟨Vorbesetzung⟩
                                  [PWL=]    ⟨Paßwort⟩
                                  [PWLS=]   ⟨Paßwort⟩

⟨operatorspezifischer
 Gebietsname⟩              ::= ⟨Name⟩

⟨prozeßspezifischer
 Gebietsname⟩              ::= ⟨Name⟩

⟨Lebensdauer⟩              ::= L|D

⟨Gebiets-Adressierungs-
 bedingung⟩                ::= { K|V
                                 B|BV
                                 D|DK
                                 F|FV } |N

⟨Verarbeitungsklasse⟩      ::= KS|MK|TR|PL

⟨Lagerklasse⟩              ::= MK|TR|PL|HG

⟨Vorbesetzung⟩             ::= (⟨Typenkennung⟩ , '⟨Tetrade⟩^12')

⟨Typenkennung⟩             ::= 0|1|2|3

⟨Paßwort⟩                  ::= ⟨Name⟩|0
```

Explizit vereinbarte Gebiete (durch den GEBIET-Befehl) müssen explizit im Adressenraum angeordnet werden (durch den GEBAN-Befehl, siehe C.6.9). Ein GEBIET-Befehl ist nur sinnvoll, wenn dem Gebiet auch Zonen (durch den ZONAN-Befehl, siehe C.6.7) zugeordnet werden.

Durch den GEBIET-Befehl wird ein Gebiet explizit vereinbart. Die Gebietsparameter können in beliebiger Reihenfolge geschrieben werden. Wird allerdings der Name mit dem Gleichheitszeichen fortgelassen, so wird, ausgehend von den unbenannten Parametern oder vom Anfang der Parameterliste, die obenstehende Reihenfolge vorausgesetzt. Dadurch ist eine Mehrfachbesetzung eines Parameters möglich. In einem solchen Fall ist der letzte Parameterwert gültig und es wird eine Warnung ausgegeben.

Die Gebietsparameter bedeuten im einzelnen:

ONAME

Der operatorspezifische Gebietsname muß eindeutig sein in allen zum Operator gehörigen Montageobjekten.

PNAME

Zusätzlich zum operatorspezifischen Gebietsnamen im Benennungsteil kann auch ein prozeßspezifischer Gebietsname angegeben werden. Dieser muß dann in dem Prozeß, zu dem der Operator gehört, eindeutig sein.
Bei ONAME und PNAME wirken nur die ersten 6 Zeichen unterscheidend.
Mindestens einer der beiden Namen muß angegeben werden.

LD

gibt die Lebensdauer des Gebiets an. L heiß "Laufzeitgebiet", D "Dauergebiet".
Der Parameter ist mit D vorbesetzt.

ADR

stellt eine Adressierungsbedingung. Die Bedeutung der Adressierungsbedingung ist, auf Gebiete übertragen, dieselbe wie bei Adressenzonen (siehe C. 5. 2). Zu ADR = B und ADR = BV ist eine Erläuterung angebracht. Alle Gebiete mit ADR = B oder ADR = BV sollen in dieselbe Großseite, und zwar in die Großseite, in der die B-Adressenzonen liegen, falls es solche gibt.

Die Angabe N bedeutet, daß das Gebiet nicht adressiert ist (Hintergrundgebiet).

Der Parameter ist mit KS vorbesetzt.

VK1 und VK2

geben die gewünschte und die geforderte Verarbeitungsklasse an.
Es bedeutet:

KS : Kernspeicher
MK : Massenkernspeicher
TR : Trommel
PL : Platte

Beide Parameter sind mit KS vorbesetzt.

LK

gibt die Lagerklasse an. Es bedeutet:

HG : Hintergrundspeicher

Die Bedeutung der anderen Werte ist wie bei VK1 und VK2.

Der Parameter ist mit PL vorbesetzt.

VB

Mit diesem Parameter kann ein Vorbesetzungsmuster angegeben werden. Durch ⟨Typenkennung⟩ wird die Typenkennung und durch die Tetradenfolge das Bitmuster eines jeden Ganzwortes in der Vorbesetzung angegeben. In dem Gebiet wird das allgemeine Vorbesetzungsmuster (siehe C. 11. 4) mit dem angegebenen Muster überschrieben.

PWL

Dieser Parameter stellt eine Bedingung für die Beschaffung des Gebiets als Fremdgebiet ("Paßwort für die Beschaffung zum Lesen"). Fehlt der Parameter, so kann sich jeder Prozeß das Gebiet zum Lesen beschaffen. Ist er mit einem Paßwort angegeben, so wird das Gebiet nur bei Anlieferung des Paßwortes als Fremdgebiet zum Lesen freigegeben. In diesem Fall muß auch der Parameter PWLS angegeben sein. Steht statt des Paßworts die Ziffer 0, so ist das Gebiet nicht als Fremdgebiet beschaffbar. Es muß dann auch PWLS = 0 sein.

PWLS

Dieser Parameter stellt eine Bedingung für die Beschaffung des Gebiets als Fremdgebiet ("Paßwort für die Beschaffung zum Lesen und Schreiben"). Fehlt der Parameter, so kann sich jeder Prozeß das Gebiet zum Lesen und Schreiben beschaffen. Ist er mit einem Paßwort angegeben, so wird das Gebiet nur bei Anlieferung des Paßworts zum Lesen und Schreiben freigegeben. Steht statt des Paßworts die Ziffer 0, so ist das Gebiet als Fremdgebiet zum Lesen und Schreiben nicht beschaffbar.

5.7. LUECKE-Befehl

```
⟨LUECKE-Befehl⟩  ::=  LUECKE (⟨Lückenname⟩ [ ,⟨Gebiets-Adressierungsbedingung⟩])
⟨Lückenname⟩     ::=  ⟨Name⟩
```

Die durch den LUECKE-Befehl vereinbarten Lücken müssen explizit im Adressenraum angeordnet werden (GEBAN-Bfehl, siehe C. 6. 9). Ein LUECKE-Befehl ist nur sinnvoll, wenn der Lücke Freihaltezonen (durch ZONAN-Befehl, siehe C. 6. 7) zugeordnet werden.

Durch den LUECKE-Befehl wird eine Lücke im Adressenraum vereinbart und mit einem Namen versehen. Eine Lücke beginnt mit einer Seite und umfaßt ein Vielfaches einer Seite.

Der Lücke wird zu Beginn des Operatorlaufs kein physikalischer Speicher zugewiesen. Sie verhält sich zum Gebiet wie die Freihaltezone zu den anderen Zonen. In einer Lücke können auch nur Freihaltezonen angeordnet werden und alle Freihaltezonen werden in Lücken angeordnet (siehe ZONAN-Befehl C. 6. 7).

Die Größe der Lücke ergibt sich aus der Gesamtheit der ihr zugeordneten Freihaltezonen.

Die Adressierungsbedingung entspricht der Bedeutung im GEBIET-Befehl (siehe C. 5. 6). Fehlt sie, so wird D angenommen.

6. Anordnung von Informationseinheiten, Adressenzonen und Gebieten

6. 1. Verteilung der Informationseinheiten auf die Adressenzonen

Jede Informationseinheit (die meisten Pseudobefehle ausgenommen) besitzt implizit einen Ablageschlüssel (K, V, B oder D) und einen Literalschlüssel (0, 1, 2 oder G).

Der implizite Ablageschlüssel ist

- K für Konstanten (B. 10),
- V für Arbeitsspeicher (C. 4. 1),
- B für Befehle (B. 7) und
- D für Datenspeicher (C. 4. 2).

Der implizite Literalschlüssel ist

0 für Informationseinheiten außerhalb von Literalen
1 für Informationseinheiten in Literalen erster Ordnung (B. 8)
2 für Informationseinheiten in Literalen zweiter Ordnung (B. 8) und
G für alleinstehende Ganzwortkonstanten in Literalen (Ganzwortliterale)

Ablageschlüssel und Literalschlüssel bilden zusammen die Ablagekennung. Folgende Ablagekennungen sind möglich

K0	V0	B0	D0
K1	V1	B1	D1
K2	V2	B2	D2
KG			

Der implizite Ablageschlüssel kann explizit verändert werden, wobei sich dann auch die Ablagekennung entsprechend verändert. Der Literalschlüssel ist im allgemeinen unveränderlich. Lediglich im Falle der Ablagekennung KG wird der Literalschlüssel notwendigerweise in 0, 1 oder 2 sinngemäß abgeändert, wenn der Ablageschlüssel explizit verändert wird.

Die explizite Veränderung des Ablageschlüssels kann durch Angabe einer Ablagespezifikation (Kapitel B. 6. 1) oder durch den nachfolgend beschriebenen STARR-Befehl (Kapitel C. 6. 2) vorgenommen werden. Während sich die Spezifikation stets nur auf eine Informationseinheit bezieht, hat der STARR-Befehl die Wirkung einer Voreinstellung, die bis auf Widerruf gültig ist. Eine Ablagespezifikation im Wirkungsbereich eines STARR-Befehls dominiert über diesen.

Der Assembler verteilt nun aufgrund der Ablagekennung die Informationseinheiten in der Reihenfolge ihrer Bearbeitung auf die verschiedenen Adressenzonen. Wird explizit nicht anders verfügt, so wird jede Informationseinheit in derjenigen implizit definierten Adressenzone ablegt, deren Bezeichnung mit der Ablagekennung übereinstimmt (siehe Kapitel C. 5. 1).

Durch den ABLAGE-Befehl (Kapitel C. 6. 4) kann die Ablage jedoch auf eine explizit definierende Adressenzone umgesteuert werden. Dabei können gleichzeitig mehrere Ablagekennungen für unterscheidbar erklärt werden. Die Wirkung des ABLAGE-Befehls kann auf Teile der Quelle beschränkt werden.

Eine schematische Übersicht befindet sich in Kapitel C. 6. 10.

6.2. STARR-Befehl

```
⟨STARR-Befehl⟩          ::= STARR ␣ ⟨Ablagespezifikation⟩
⟨Ablagespezifikation⟩ ::= K|V|B|D
```

Der STARR-Befehl ordnet allen folgenden Informationseinheiten den in der Ablagespezifikation angegebenen Ablageschlüssel zu (siehe C. 6. 1).
Die Wirkung des STARR-Befehls wird entweder durch den nächsten STARR-Befehl oder durch einen STEND-Befehl aufgehoben.
Eine Ablagespezifikation im Spezifikationsteil einer Informationseinheit überspielt für diese Informationseinheit die Wirkung des STARR-Befehls.

Folgen ein benannter SEGM-Befehl und ein STARR-Befehl in dieser Reihenfolge unmittelbar aufeinander, so ist zu beachten, daß im allgemeinen der Name des Segments nicht einer Informationseinheit derselben Zone zugeordnet wird, in der die folgenden Informationseinheiten angeordnet sind. Dies kann dadurch umgangen werden, daß man den STARR-Befehl vor den benannten SEGM-Befehl legt.

6.3. STEND-Befehl

```
⟨STEND-Befehl⟩ ::= STEND
```

Der STEND-Befehl beendet den Wirkungsbereich eines vorangehenden STARR-Befehls.

6.4. ABLAGE-Befehl


```
⟨ABLAGE-Befehl⟩   ::= ABLAGE ␣ ⟨Zonenname⟩ ([⟨Ablagekennung⟩])
⟨Zonenname⟩       ::= ⟨Name⟩
⟨Ablagekennung⟩   ::= { K0 | V0 | B0 | D0 }
                      { K1 | V1 | B1 | D1 }
                      { K2 | V2 | B2 | D2 }
```

Alle Informationseinheiten nach dem ABLAGE-Befehl, die explizit (durch STARR-Befehl oder Spezifikation), bzw. implizit eine der Ablagekennungen bekommen, die im ABLAGE-Befehl aufgeführt sind, werden der im ABLAGE-Befehl genannten Zone zugeordnet.

Sind schon aufgrund früherer ABLAGE-Befehle derselben Adressenzone Ablagekennungen zugeordnet, so gelten außer den neuen Zuordnungen auch noch die bereits bestehenden.

Mit dem AEND-Befehl (siehe C. 6. 5) können die Zuordnungen wieder auf die implizite Einstellung (siehe C. 6. 1) zurückgeschaltet werden.

Informationseinheiten, deren Ablagekennungen durch einen ABLAGE-Befehl einer Adressenzone zugeordnet sind, werden in der Reihenfolge ihrer Bearbeitung in dieser Adressenzone abgelegt. Die Adressenzone muß vor dem ABLAGE-Befehl definiert sein (siehe C. 5. 2, C. 5. 3, C. 5. 4).

Anmerkung: Es wird darauf hingewiesen, daß durch den ABLAGE- und AEND-Befehl lediglich die Zuordnung von Ablagekennungen zu den Adressenzonen verändert wird. Die bis dahin bereits abgelegten Informationseinheiten sind selbstverständlich davon in keiner Weise betroffen.

6.5. AEND-Befehl

```
⟨AEND-Befehl⟩ ::= AEND ([⟨Ablagekennung⟩])
```

Durch den AEND-Befehl wird die (eventuelle) Zuordnung der aufgeführten Ablagekennungen zu explizit definierten Adressenzonen aufgehoben. Informationseinheiten mit diesen Ablagekennungen werden dann wieder in den zugehörigen implizit definierten Adressenzonen abgelegt.

6.6. Anordnung von Adressenzonen in Gebieten

Der Montierer erstellt für die implizit definierten Adressenzonen Gebiete. Solche Gebiete heißen "implizit definiert".

Die explizit definierten Adressenzonen (siehe C. 5. 1) werden genauso wie die implizit definierten behandelt, wenn über ihre Anordnung in einem explizit definierten Gebiet nicht durch einen ZONAN-Befehl explizit verfügt wird.

Freihaltezonen werden nicht in Gebieten, sondern in Lücken angeordnet.

6.7. ZONAN-Befehl

```
⟨ZØNAN-Befehl⟩ ::= ZØNAN ␣ { ⟨Gebietsname⟩ | ⟨Lückenname⟩ } ([⟨Zonenname⟩] )
⟨Gebietsname⟩  ::= ⟨Name⟩
⟨Lückenname⟩   ::= ⟨Name⟩
⟨Zonenname⟩    ::= ⟨Name⟩
```

Alle in einem ZONAN-Befehl auftretenden Zonen und das Gebiet bzw. die Lücken müssen vorher im Quellenprogramm deklariert sein (durch ZONE- bzw. GEBIET-Befehl usw.).

Steht vor der Klammer ein Gebietsname, so dürfen in der Klammer keine Namen von Freihaltezonen vorkommen. Die aufgeführten Adressenzonen werden dann in der angegebenen Reihenfolge unter Berücksichtigung eventuell vorhandener Anfangsbedingungen unmittelbar hintereinander in dem genannten Gebiet angeordnet.

Jede vom Assembler erzeugte Zone beginnt auf einer geraden Adresse ; stärkere Bedingungen resultieren aus Teilseitenangaben bei Arbeitsspeichern (siehe C. 3).

Es können mehrere ZONAN-Befehle mit dem gleichen Gebietsnamen auftreten. Die Adressenzonen werden dann fortlaufend abgelegt in der durch die Bearbeitung der ZONAN-Befehle bestimmten Reihenfolge.

Verlangt die Adressierungsbedingung einer Zone Schreibschutz, die Adressierungsbedingung des Gebiets jedoch nicht, so wird vom Montierer eine Warnung ausgegeben und die Adressierungsbedingung der Zone entsprechend abgewandelt, d.h. es wird

K in V,
V in K,
B in BV,
D in DK,
BV in B,
F in FV,
FV in F,
DK in D

abgeändert.

Steht vor der Klammer ein Lückenname, so dürfen in der Klammer nur Namen von Freihaltezonen vorkommen. Das übrige gilt dann sinngemäß.

6.8. Anordnung von Gebieten im Adressenraum

Explizit vereinbarte Gebiete (siehe C. 5. 6) und Lücken (siehe C. 5. 7) müssen explizit im Adressenraum angeordnet werden. Das geschieht durch den in Kapitel C. 6. 9 beschriebenen GEBAN-Befehl.

Da in diesem Zusammenhang Gebiete und Lücken als vollkommen gleichartig betrachtet werden, gilt das folgende auch für Lücken, obwohl nur von Gebieten die Rede ist.

Die Lage eines Gebiets im Adressenraum kann durch eine Absolutadresse oder durch eine Reihenfolgeangabe festgelegt werden. Jedem Gebiet, für das keine Absolutadresse gegeben wird, muß durch die Reihenfolgeangabe eindeutig ein unmittelbarer Vorgänger zugeordnet sein.

Ein Gebiet mit der Angabe einer Absolutadresse darf keinen Vorgänger in der Reihenfolgefestlegung haben.

Durch die Anordnung der Gebiete im Adressenraum wird jedem von ihnen direkt (Absolutadresse) oder indirekt (Reihenfolgeangabe) ein zusammenhängender Teil des Adressenraums zugewiesen. Steht dieser im Widerspruch zu der Adressierungsbedingung des Gebiets (z.B. ADR = K und Absolutadresse ist größer als 2^{16}), so wird die Montage abgebrochen.

Bei der Adressenraumzuteilung an ein Gebiet wird indirekt allen im Gebiet angeordneten Adressenzonen ebenfalls Adressenraum zugewiesen. Dies kann eine Verletzung der Adressierungsbedingung der Zone (siehe C.5.1, C.5.2) zur Folge haben. In diesem Fall wird gleichfalls die Montage abgebrochen.

Man beachte, daß die Adressierungsbedingungen der Zonen nicht mit der Adressierungsbedingung des Gebiets übereinzustimmen brauchen. Z.B. kann die Zone die Adressierungsbedingung F haben und für das Gebiet ADR = DK sein. Die Bedingungen sind dann schon erfüllt, wenn nur die Zone nicht über eine Großseitengrenze geht.

6.9. GEBAN-Befehl

⟨GEBAN-Befehl⟩ ::= GEBAN ␣ ⟨Bezugsparameter⟩ ({ ⟨Gebietsname⟩ / ⟨Lückenname⟩ }ⁿ)

⟨Bezugsparameter⟩ ::= { ⟨Seitenadresse⟩ / ⟨Gebietsname⟩ / ⟨Lückenname⟩ }

⟨Seitenadresse⟩ ::= ⟨Zahl⟩ | ⟨Tetradenfolge⟩

Alle in einem GEBAN-Befehl genannten Gebiete und Lücken müssen vorher im Quellenprogramm deklariert sein (GEBIET, bzw. LUECKE-Befehl).

Alle GEBAN-Befehle, die sich auf einen Operator beziehen, bestimmen zusammen die Anordnung der explizit vereinbarten Gebiete im Adressenraum.

Durch den Bezugsparameter ist eine bestimmte absolute Seitenadresse gegeben, entweder direkt als Dezimal- oder Sedezimalzahl oder indirekt als Adresse der ersten Seite nach dem angegebenen Gebiet oder Lücke.
Im Fall einer Zahlenangabe bedeutet diese die durch 2048 geteilte Adresse des ersten Halbworts der Seite.

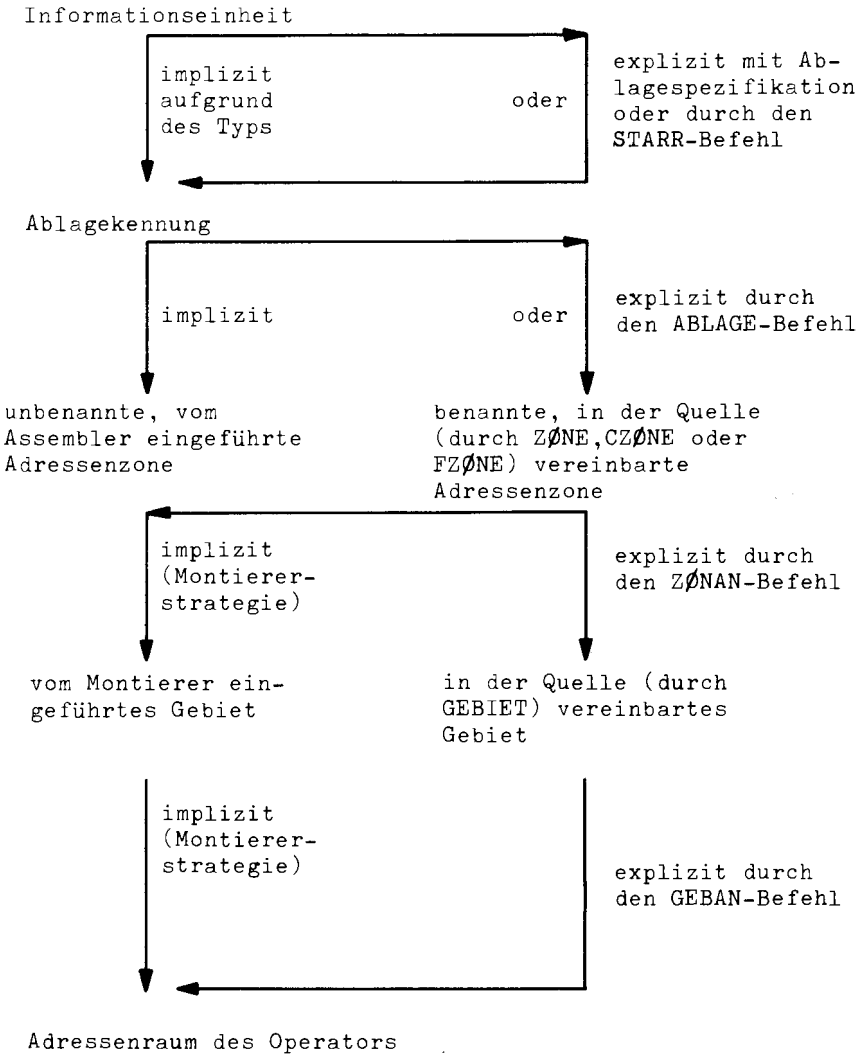
Der Bezugsparameter kann auch ein in einem anderen Montageobjekt für den gleichen Operator definiertes Gebiet bzw. eine solche Lücke nennen.

Die in der Klammer genannten Gebiete und Lücken werden ab der durch den Bezugsparameter gegebenen Seitenadresse im Adressenraum in dichter Folge angeordnet.

Aus den Vorschriften für die Anordnung der Gebiete (siehe C. 6. 8) ergeben sich folgende formale Forderungen an die Parameter des GEBAN-Befehls:

- Jeder Gebiets- und Lückenname muß in allen zu einem Operator gehörigen GEBAN-Befehlen genau einmal vorkommen.
- Der Bezugsparameter des GEBAN-Befehls, in dem ein Gebietsname in der Klammer vorkommt, soll "der Bezugsparameter" des Gebiets heißen. Ebenso gibt es den Bezugsparameter einer Lücke.
- Wenn der Bezugsparameter eines Gebiets oder einer Lücke wieder ein Gebiets- oder Lückenname ist, so kann man wiederum dessen Bezugsparameter bestimmen. Durch fortgesetztes Aufsuchen des Bezugsparameters muß man schließlich zu einer Seitenadresse gelangen.

6.10. **Schematische Übersicht für die Anordnung von Informationseinheiten, Adressenzonen und Gebieten.**



7. Querbezüge zwischen Quellenprogrammen

Bei dem auf den Übersetzungsvorgang folgenden Montagevorgang können mehrere übersetzte Quellenprogramme zu einem ablauffähigen Maschinenprogramm vereinigt werden. Daher darf in Quellenprogrammen auf bestimmte Adressen anderer Quellenprogramme, die später nach der Übersetzung angeschlossen werden, Bezug genommen werden. Diese Querbezüge erfolgen in der Quellensprache mit Hilfe der diesen zugeordneten Namen. Diese Namen müssen sowohl in dem Quellenprogramm, in dem sie definiert sind, als auch in dem Quellenprogramm, das sie verwendet, durch eigene Pseudobefehle vereinbart werden. Dabei werden zwei Arten von Namen unterschieden : Montageobjektnamen und Kontaktnamen.

7.1. Montageobjektnamen und Kontaktnamen

Montageobjekte sind übersetzte Quellenprogramme, die für einen Montagevorgang bereitstehen. Ein Montageobjekt kann mehrere Namen haben. Im UEBERSETZE-Kommando (siehe Kommandohandbuch) kann höchstens ein Montageobjektname (abgekürzt: MO-Name) angegeben werden. Wird im UEBERSETZE-Kommando kein MO-Name angegeben, so wird der Name des Quellenprogramms MO-Name (siehe C. 3. 1). Ist das Quellenprogramm unbenannt, so erhält das Montageobjekt den Namen STDHP.

Mit dem EINGG-Befehl können zusätzlich beliebig viele MO-Namen vereinbart werden.

Kontaktnamen sind Namen, die zwischen Montageobjekten bekannt sind und denen Kernspeicheradressen zugeordnet sind. Es lassen sich also mit ihnen Querverbindungen zwischen Quellenprogrammen herstellen. Kontaktnamen sind genau die in EINGG-Befehlen genannten Namen. Ein Kontaktname muß in dem Quellenprogramm, in dem er in einem EINGG-Befehl erscheint, als symbolische Speicheradresse definiert sein.

Um die Montage zu ermöglichen, muß in jedem, beim Montagevorgang benötigten Programm angegeben sein

- mit welchen Namen sich ein fremdes Programm auf Adressen des eigenen Programms beziehen kann. Diese Angabe erfolgt in der TAS-Quellensprache mittels eines oder mehrerer EINGG-Befehle.

- mit welchen Kontakt- und Montageobjektnamen sich das eigene Quellenprogramm auf Adressen fremder Quellenprogramme beziehen kann. Diese Angabe erfolgt in der TAS-Quellensprache mittels eines oder mehrer EXTERN-Befehle.

7.2. EXTERN-Befehl

⟨EXTERN-Befehl⟩ ::= EXTERN ␣ ⟨fremder MØ-Name⟩ [({⟨fremder Kontaktname⟩}+)]

Mit Hilfe eines EXTERN-Befehls kann ein fremder MO-Name und zusätzlich eine Reihe von fremden Kontaktnamen angegeben werden. Diese Kontaktnamen müssen jedoch in dem angegebenen fremden Montageobjekt als Kontaktnamen definiert sein. Selbstverständlich muß bei der Montage dieses Montageobjekt zur Verfügung stehen. Die fremden Kontaktnamen können im ganzen Quellenprogramm wie normale globale Namen von Kernspeicheradressen benutzt werden. Jedoch darf in einer Informationseinheit nicht Bezug auf mehr als einen fremden Kontaktnamen genommen werden.

Auf (Ganz-) Adressen, die einen Externbezug enthalten, wird bei der Montage die Absolutadresse des Kontaktnamens addiert (der Fremdname wird zunächst (für den Montagecode) in eine Null übersetzt).

Ein fremder MO-Name kann nur dann wie der Name einer Kernspeicheradresse benutzt werden, wenn er beim EXTERN-Befehl auch als Kontaktname aufgeführt wird (3. Beispiel) oder wenn der EXTERN-Befehl nur den fremden MO-Namen enthält (1. Beispiel). Der fremde MO-Name muß hierzu im fremden Montageobjekt als Kontaktname vereinbart sein (durch EINGG).

Beispiele

```
EXTERN    ANTON,
EXTERN    ZMO(A,B,C),
EXTERN    FRMO  (FRMO,D,E,F),
```

7.3. EXTOPT-Befehl

```
⟨EXTØPT-Befehl⟩  ::=  EXTØPT ␣ ⟨fremder MØ-Name⟩ [({⟨fremder Kontaktname⟩})]
```

Mit Hilfe eines EXTOPT-Befehls können eine Reihe fremder Kontaktnamen angegeben werden. Es werden die gleichen Kontaktdeklarationen erzeugt wie beim EXTERN-Befehl. Im Gegensatz zum EXTERN-Befehl muß jedoch das fremde Montageobjekt bei der Montage nicht zur Verfügung stehen, da es nicht notwendig anmontiert wird.

Fremde Montageobjekte, die im Adreßteil von EXTOPT-Befehlen aufgeführt sind, werden nur dann anmontiert, wenn dieses explizit verlangt wird (durch das MONTIERE-Kommando, oder durch einen EXTERN-Befehl in einem anderen anzumontierenden Montageobjekt).

7.4. EINGG-Befehl

```
⟨EINGG-Befehl⟩  ::= EINGG { ⟨zusätzlicher eigener MØ-Name⟩ }
                          { ({⟨eigener Kontaktname⟩}) }
```

Durch einen EINGG-Befehl kann entweder ein zusätzlicher eigener MO-Name oder eine Liste eigener Kontaktnamen vereinbart werden.

Jeder zusätzliche eigene MO-Name und jeder eigene Kontaktname muß als Name einer Kernspeicherzelle des eigenen Programms vereinbart sein (d.h. als Benennung einer Konstanten, eines Befehls oder eines Speicherbereichs auftreten). Die Verwendung von gleichgesetzten Namen ist nicht erlaubt.

Ein zusätzlicher eigener MO-Name (sogen. unechter MO-Name) wird gleichzeitig auch Kontaktname (kann von fremden Montageobjekten als Kernspeicheradresse benutzt werden).

Ein doppeltes Komma in der Namensliste eines EINGG-Befehles wirkt wie das Ende der Informationseinheit. Die folgenden Namen in der Liste werden bei der Übersetzung ohne Fehlermeldung überlesen. Tritt zur Montagezeit die Fehlermeldung "Eingangsname ... im Montageobjekt ... nicht deklariert" auf, so ist als Grund für diese Meldung der oben beschriebene Sachverhalt in Betracht zu ziehen.

Beispiele

```
EINGG     ZMO,          zuätzlicher eigener MØ-Name
EINGG     (A,B,C),      eigene Kontaktnamen
```

Beispiel für die Benutzung des eigenen MO-Namens als Kontaktname

```
◇ UEBERSETZE,SPRACHE=TAS,NUMERIERUNG=-STD-,MO=-STD-,QUELLE=/
        FRMO=    SEGM,
                 BA     1,
                 .
                 .
                 .
                 EINGG  (FRMO,D,E,F),
                 .
                 .
                 .
```

8. Zuordnung der Indexnamen

8.1. Indexzellen und ihre Verwendung

Unter den Indexzellen versteht man bestimmte Halbworte des TR 440-Kernspeichers, die mittels der Indexbefehle angesprochen werden können.
Die insgesamt 256 Indexzellen liegen hintereinander in einem bestimmten Teil des Kernspeichers, dem sogenannten Indexspeicher. Diesen Indexzellen sind der Reihe nach die Indexadressen 0 bis 255 zugeordnet. Die Indexadresse ist also eine natürliche Zahl zwischen 0 und 255 und gibt die Lage einer Indexzelle relativ zum Beginn des Indexspeichers an (die Indexadressen können daher intern in einem halben Adressenteil eines Indexbefehls untergebracht werden).
Die Anfangsadresse des Indexspeichers wird Indexbasis genannt; sie ist im sogenannten Indexbasisregister enthalten. Ein 256 Halbworte umfassender Teil des Kernspeichers wird durch Eintragung seiner Anfangsadresse ins Indexbasisregister zum Indexspeicher. Während eines Programmlaufs kann der Inhalt dieses Registers (mit Hilfe der Befehle BCI, ZI) geändert werden. Damit können beliebige Teile des Kernspeichers zum Indexspeicher gemacht und mit Indexbefehlen adressiert werden. Welche Zelle des Kernspeichers mit Hilfe einer bestimmten Indexadresse angesprochen wird, richtet sich daher nach der Lage der Indexbasis.

Die Indexzellen können auch mit Hilfe von Ganzadressen adressiert werden. Jedoch sollte unbedingt darauf verzichtet werden, da durch die spezielle hardware-seitige Behandlung von Indexspeicherinhalten in Assoziativregistern unerwünschte Effekte eintreten können.

8.2. Benennung von Indexadressen

Indexadressen können explizit oder implizit mit Namen versehen werden. Die implizite Benennung von Indexzellen wird vom Assembler bei der Übersetzung von Indexbezügen vorgenommen. Und zwar ordnet der Assembler bei der Übersetzung solcher Bezüge der Reihe nach alle nicht als Kernspeicheradressen oder Indexadressennamen vereinbarten Namen einzelnen Indexadressen zu. Die erste auf diese Art und Weise implizit benannte Indexadresse ist um eins höher als die größte explizit vergebene Indexadresse. Die mit implizit vereinbarten Namen adressierbaren Indexzellen liegen also unmittelbar hinter den Indexzellen, die mit explizit vereinbarten Namen adressiert werden können; damit wird eine unerwünschte Mehrfachbelegung von Indexzellen verhindert.
Die implizite Benennung von Indexadressen ist sinnvoll, wenn vom Programmierer für bestimmte Zwecke Indexzellen, deren Reihenfolge oder Adresse keine Rolle spielt, benötigt werden. Alle implizit vereinbarten Namen haben automatisch einen globalen Geltungsbereich. Lokale Indexadressennamen müssen daher explizit mit Hilfe des INDEX-Befehls vereinbart werden.

8.3. INDEX-Befehl

⟨INDEX-Befehl⟩	::=	INDEX ␣ [⟨Indexpegel⟩] ⟨Indexliste⟩
⟨Indexpegel⟩	::=	⟨Absolutbezug⟩
⟨Indexliste⟩	::=	({ ⟨Indexbenennung⟩ / ⟨Freihalteangabe⟩ })
⟨Indexbenennung⟩	::=	{ ⟨lokale Indexbenennung⟩ / ⟨globale Indexbenennung⟩ }
⟨lokale Indexbenennung⟩	::=	⟨Indexname⟩
⟨globale Indexbenennung⟩	::=	⟨Indexname⟩.
⟨Freihalteangabe⟩	::=	⟨Zahl⟩
⟨Absolutbezug⟩		siehe Kapitel B.7.3

Mit Hilfe des INDEX-Befehls kann man einzelnen Indexadressen bestimmte lokale und globale Indexnamen explizit zuordnen. Grundlage für diese Zuordnung ist der Indexpegel, der im Adressenteil des INDEX-Befehls enthalten ist. Der Indexpegel ist eine Indexadresse, die entweder absolut oder durch einen bereits explizit vereinbarten Indexnamen angegeben werden kann. Als selbstdefinierende Ausdrücke sind jedoch nur Zahlen zugelassen. Außerdem müssen alle im Indexpegel vorkommenden Namen bereits definiert sein. Die Ganzadresse muß hier einen Wert zwischen 0 und 255 haben. Hinter dem Indexpegel steht eine Indexliste. Die in dieser Liste enthaltenen lokalen und globalen Indexnamen werden der Reihe nach den Indexadressen zugeordnet, die auf den Indexpegel folgen. Der erste in dieser Liste enthaltene Indexname wird dabei dem Indexpegel zugeordnet. Fehlt die Indexpegelangabe, so wird die Liste vom Assembler als Verlängerung der im vorangehenden INDEX-Befehl angegebenen Indexliste interpretiert (fehlt der Indexpegel im ersten INDEX-Befehl, so wird als Indexpegel die Indexadresse 0 genommen). In der Indexliste enthaltene globale Indexnamen müssen durch einen Punkt gekennzeichnet sein.

Neben den Indexnamen darf die Indexliste auch Freihaltezahlen enthalten. Diese geben die Zahl der freizuhaltenden Indexadressen an, denen also durch den aktuellen INDEX-Befehl kein Name zugeordnet wird. Der auf eine Freihaltezahl folgende Indexname wird daher nicht der nächsten Indexadresse zugeordnet, sondern der Indexadresse, die sich aus der nächsten Indexadresse durch Addition der Freihaltezahl ergibt.

Beispiele

```
INDEX    (A.,B,2,C,D,10),
INDEX    2(E,F.),
INDEX    (G),
```

ergibt folgende Belegung:

Indexadresse	0	A
	1	B
	2	E
	3	F
	4	C und G
	5	D

Der erste implizite Indexname läge auf Adresse:

```
                INDEX   10(X),
SP1=            ASP     10,
SP2=            ASP     2,
                INDEX   X+2+SP2-SP1(Y,Z),
```

C

9. Vereinbarung von Oktadennamen

9.1. TEXT-Befehl

```
⟨TEXT-Befehl⟩    ::=   ⟨Oktadenname⟩ = TEXT ␣ ⟨Oktadenwert⟩
⟨Oktadenname⟩    ::=   ⟨Name⟩
⟨Oktadenwert⟩    ::=   ⟨Zahl⟩
```

Mit Hilfe der TEXT-Befehle kann man beliebige Oktaden mit sogenannten Oktadennamen benennen. Oktadennamen dürfen nach ihrer Vereinbarung in Textfolgen (siehe Kapitel B.4) anstelle der Oktadenwerte stehen.

Beispiele

```
E=          TEXT  228,
L=          TEXT  235,
F=          TEXT  229,
U=          TEXT  244,
N=          TEXT  237,
K=          TEXT  234,
            '(''T'',E,L,E,F,U,N,K,E,N)',
```

10. Steuerung des TAS- Protokolls

Im UEBERSETZE-Kommando kann ein Protokoll des Übersetzungsvorgangs gefordert werden. Mit Hilfe des DRUCK-Befehls kann die Protokollierung von Teilen des Quellenprogramms unterdrückt werden. Ferner kann durch die Pseudobefehle ZEILE, UVB und NOTE weiterer Einfluß auf die Gestaltung des TAS-Protokolls genommen werden.

10.1 DRUCK-Befehl

```
⟨DRUCK-Befehl⟩      ::=  DRUCK ␣ ⟨Druckparameter⟩
⟨Druckparameter⟩    ::=  0|1|2|3
```

Der DRUCK-Befehl ist eine Anweisung an den Assembler, die die Protokollierung des Quellentextes steuert. Die Art der Steuerung wird durch den im DRUCK-Befehl enthaltenen Druckparameter angegeben. Der Druckparameter kann die Werte 0, 1, 2 und 3 annehmen.
Diese Werte haben folgende Bedeutung:

0: keine Protokollierung

1: Protokollierung des originalen Quellentextes (ohne abgelehnte Versionen)

2: nur der originale Quellentext, vom Assembler erzeugte Informationseinheiten , eingeschobene Halbworte und Arbeitsspeicher werden protokolliert

3: alles wird protokolliert, d.h. gegenüber 2 auch die aus Makros erzeugten Informationseinheiten sowie die abgelehnten Versionen

Der DRUCK-Befehl kann an beliebigen Stellen des Quellenprogramms gegeben werden und bezieht sich immer auf die Protokollierung der auf den DRUCK-Befehl folgenden Informationseinheiten. Er ist immer bis zum nächsten DRUCK-Befehl bzw. bis zum ENDE-Befehl gültig. Zu Programmbeginn wird implizit der DRUCK-Befehl: DRUCK 2, gegeben. DRUCK-Befehle werden überlesen, wenn durch das UEBERSETZE-Kommando die Protokollierung des gesamten Quellenprogramms unterdrückt wird. Befehle und Konstanten, bei deren Übersetzung Fehler erkannt werden (z.B. Adressenteil fehlt, Name doppelt definiert, usw.) werden allerdings in jedem Fall mit der zugehörigen Fehlermeldung protokolliert.

Beispiele

```
NEU=    SEGM,
          .
          .
          .
        DRUCK   0,
ALT=    SEGM,
          .
          .
          .
```

10.2. ZEILE-Befehl

```
⟨ZEILE-Befehl⟩ ::= ZEILE ␣ ⟨Zeilenzahl⟩
⟨Zeilenzahl⟩   ::= 1|2|3|4|5|6
```

Im Assemblerprotokoll wird der ZEILE-Befehl durch entsprechend viele Leerzeilen ersetzt.

10.3. UVB-Befehl

```
⟨UVB-Befehl⟩ ::= UVB { 0 }
                     { 1 }
```

Der UVB-Befehl dient zur Unterdrückung der Fehlermeldung "verbotener Bezug". Die Wirkung wird mit 1 für die folgenden Einheiten eingeschaltet und mit 0 wieder ausgeschaltet.

Beispiel für sinnvolle Anwendung

```
          UVB     1,
EINGABE=  S       NZEIN1/K      --NAECHST ZEICH 1.ART--
          S       NZEIN2/K      --NAECHST ZEICH 2.ART--
          MU      S     0/K     --EINGABE-ENDE--
          UVB     0,
          MF      XEIN,
          T       EINGABE,
```

10.4. NOTE-Befehl

```
⟨NØTE-Befehl⟩  ::=  NØTE ␣ ⟨Oktadenfolge⟩
```

Falls der Assembler den NOTE-Befehl interpretiert, er z.B. nicht in einer unterdrückten Versionsfolge liegt, wird der angegebene Oktadenstring in jedem Fall im Protokoll als Kommentar, quasi mit Pseudo-Fehlermeldung, ausgedruckt.
Der NOTE-Befehl ist sinnvoll im Zusammenhang mit Makros: Z.B. zur Anzeige aktueller Makrovariablenwerte oder zur Auswahl von Versionen.

Beispiele

```
NOTE    ''AUFRUFNR. +(NUMMER*)'',
NOTE    ''AUFRUF OHNE ARGUMENT'',
```

11. Startanweisungen für Operatoren

Operatoren sind ablauffähige Maschinenprogramme, die von einem Prozeß gestartet werden und danach unter der Regie des Prozesses ablaufen (näheres siehe Beschreibung des TR 440-Betriebssystems).

Vor dem Start eines Operators werden bestimmte genau definierte Anfangsbedingungen hergestellt. Diese Anfangsbedingungen müssen mit Hilfe der sogenannten Startanweisungen angegeben werden (C.11.1 - C.11.5).

Jede dieser Startanweisungen muß genau einmal in der Gesamtquelle vorkommen. Die Ganzadressen der Startanweisungsbefehle dürfen nur symbolische Adressen sein.

In Programmen, die mit TASR (UEBERSETZE-Kommando; Rahmenprogramm einassembliert) übersetzt werden, dürfen keine Startanweisungsbefehle vorkommen.

Der STRUKT-Befehl steht im Zusammenhang mit einer eventuellen Postmortem-Behandlung

11.1. XBASIS-Befehl

```
⟨XBASIS-Befehl⟩  ::=  XBASIS ␣ ⟨Ganzadresse⟩
⟨Ganzadresse⟩        siehe Kapitel B.7.3
```

Diese Anweisung bewirkt, daß vor dem Start des Operators die Indexbasis auf die durch den Kernspeicherbezug angegebene Adresse gesetzt wird.

11.2. ALARM-Befehl

```
⟨ALARM-Befehl⟩  ::=  ALARM ␣ ⟨Ganzadresse⟩
⟨Ganzadresse⟩        siehe Kapitel B.7.3
```

Diese Anweisung bewirkt, daß vor dem Start die angegebene Adresse als Alarmadresse gesetzt wird.

Beispiele

```
ALARM    AADR,
```

11.3. UNTPR-Befehl

```
⟨UNTPR-Befehl⟩  ::=  UNTPR ␣ ⟨Indexbezug⟩
⟨Indexbezug⟩         siehe Kapitel B.7.6
```

Diese Anweisung bewirkt, daß vor dem Start der Unterprogrammordnungszähler auf die durch den Indexbezug angegebene Indexadresse gesetzt wird.

11.4. VORBES-Befehl

```
⟨VØRBES-Befehl⟩  ::=  VØRBES (⟨Typenkennung⟩, '⟨Tetrade⟩12')
⟨Typenkennung⟩   ::=  0|1|2|3
```

Diese Anweisung bewirkt, daß vor dem Start alle durch ASP und DSP vereinbarten Speicherbereiche mit dem angegebenen Muster vorbesetzt werden. Es ist jedoch auch möglich, diese allgemeine Vorbesetzung auf einzelnen explizit definierten Gebieten abzuändern (siehe C. 5. 6).

11.5. START-Befehl

```
⟨START-Befehl⟩  ::=  START ␣ ⟨Ganzadresse⟩
⟨Ganzadresse⟩        siehe Kapitel B.7.3
```

Diese Anweisung nennt die Adresse, bei der der Operatorlauf beginnen soll, wenn er gestartet wird.

11.6. STRUKT-Befehl

```
⟨STRUKT-Befehl⟩       ::=  STRUKT ( { [⟨Typ = 1⟩,] ⟨Kontrollblockname⟩ }
                                    { ⟨Typ ≠ 1⟩ [,⟨Kontrollblockname⟩] } )

⟨Kontrollblockname⟩   ::=  ⟨Name⟩

⟨Typ⟩                 ::=  { '⟨Tetrade ≠ 0⟩' }
                           { ⟨1 ≦ Zahl ≦ 15⟩ }
```

In jedem Quellenprogramm darf höchstens ein STRUKT-Befehl vorkommen. Die im STRUKT-Befehl mitgegebene Information wird nur in Testversionen im Alarmfall bei der Rückverfolgung der Unterprogrammaufrufhierarchie ausgewertet.

⟨ Typ⟩ gibt an, nach welchen Unterprogrammkonventionen das Quellenprogramm aufgebaut ist. Näheres ist den Handbüchern des Programmiersystems zu entnehmen. Der Typ 1 bedeutet, daß das Quellenprogramm ein Unterprogramm nach den allgemeinen Unterprogrammkonventionen (siehe TAS-Handbuch) ist und daß der Kontrollblock bei der angegebenen Adresse liegt.

Das Fehlen des Parameters ⟨ Typ = 1⟩ ist zwar erlaubt, führt aber zu einer Fehlermeldung (Warnung) des Assemblers.

Beispiele

```
STRUKT      (1,KONTROLLBL),
```

12. Gleichsetzung

12.1. GLCH-Befehl

⟨GLCH-Befehl⟩	::=	⟨Gl.Benennung⟩ GLCH ␣ ⟨Gl.Adressenteil⟩
⟨Gl.Benennung⟩	::=	{ ⟨Gl.Name⟩ = ⟨Gl.Name⟩. = }
⟨Gl.Name⟩	::=	⟨Name⟩
⟨Gl.Adressenteil⟩	::=	{ ⟨Ganzadresse ≠ Literal⟩ ⟨Rechtsadresse⟩ ⟨Linksadresse⟩ ⟨Linksadresse⟩␣⟨Rechtsadresse⟩ }

Mit Hilfe des GLCH-Befehls können beliebige Adressenteile oder bestimmte Teile von Adressenteilen mit einem lokalen oder globalen Gleichsetzungsnamen versehen werden. Dieser Name darf in beliebigen Adressenteilen anstelle des benannten Ausdrucks stehen. Die Gleichsetzung wird wie eine Quelltextersetzung ausgeführt (Vorsicht bei der Verwendung des Minuszeichens im Gleichsetzungstext sowie Achtung auf Segmentgrenzen (siehe Beispiel 1)). Die Gleichsetzungsnamen müssen so gewählt werden, daß bei der Verwendung des GL. Namen in Adressenteilen eine Verwechslung des GL. Namen mit anderen Ausdrücken, die im Adressenteil erlaubt sind, ausgeschlossen ist (siehe Beispiel 2).

Der GLCH-Befehl muß genau eine Benennung haben.
Gleichgesetzte Namen dürfen nicht im Adreßteil von Pseudobefehlen stehen. Ausnahme: GLCH, ASP und DSP.
Gleichsetzungen können mehrstufig sein (bis zu 16); rekursive Verweise sind verboten (da unsinnig).
Gleichsetzungen brauchen nicht vor der Stelle ihrer Anwendung definiert zu sein, ausgenommen bei ASP und DSP (siehe C. 4).

Beispiele

```
1)              SEGM,
      A=        1,
      B.=       GLCH    A,

                SEGM,
                B       B       :IST EIN FEHLER,DA A IN DIESEM
                                 SEGMENT UNBEKANNT IST;
      C=        GLCH    D-1,
      D=        ASP 2,
      E=        ASP 2,
                BA      E-C     :GLEICHBEDEUTEND MIT BA  1,;
```

```
2)    AQ=       GLCH    DH,

                RT      AQ      :HIER WIRD KEINE GLEICHSETZUNG
                                 AUSGEFUEHRT;
```

```
3)    A=        GLCH    B,
      B=        GLCH    C,
                B       A,

      X7=       GLCH    X3,
                XB      X7,

                SH      SPEZ,
      SPEZ=     GLCH    ALK   10,

      T3=       GLCH    34,
                LZL     T3    0,
                BA      T3,
```

13. Codeumsteuerung für Oktadenfolgen

13.1. CODE-Befehl

```
⟨CØDE-Befehl⟩          ::=  CØDE ␣ ⟨Codespezifikation⟩
⟨Codespezifikation⟩    ::=  D|S|Z
```

Mittels des CODE-Befehls kann der vom TAS-Assembler erzeugte Objektcode von Oktadenkonstanten und von Oktadenfolgen in Textkonstanten bzw. in Bitfeldkonstanten in einen vom Zentralcode verschiedenen Code umgeschlüsselt werden.

Durch die Codespezifikation D wird in Druckercode DC2, durch die Codespezifikation S wird in Schreibmaschinencode SMC1 umgeschlüsselt.

Zu Beginn der TAS-Übersetzung ist implizit der Zentralcode ZC1 eingestellt; ferner kann der Zentralcode durch die Codespezifikation Z eingestellt werden.

Jeder CODE-Befehl ist gültig bis zum Auftreten des nächsten oder des ENDE-Befehls.

14. Replikation von Informationseinheiten

14.1. REPL-Befehl

```
⟨Replikation⟩            ::=  ⟨REPL-Befehl⟩ ⟨Replikationsblock⟩ ⟨REND-Befehl⟩

⟨REPL-Befehl⟩            ::=  REPL ␣ [⟨Ablagebestimmung⟩] (⟨Wiederholungsangabe⟩)
                              [/⟨Spezifikation⟩]

⟨Ablagebestimmung⟩       ::=  K|V|B|D

⟨Wiederholungsangabe⟩    ::=  ⟨natürliche Zahl ≤ 65535⟩

⟨Spezifikation⟩          ::=  U|G

⟨Replikationsblock⟩      ::=  ⟨IE⟩ [⟨IE⟩] 126

⟨IE⟩                     ::=  ⟨Informationseinheit, ausgenommen die Pseudobefehle
                              ASP,DSP,ABLAGE,AEND,REPL,REND⟩
```

Durch den REPL-Befehl wird ein Replikationsblock eingeleitet, der maximal 127 Einheiten enthalten darf. Er bewirkt, daß der Replikationsblock n-mal (der Wiederholungsangabe entsprechend) in eine bestimmte Zone abgelegt wird. Die Ablage kann durch die Ablagespezifikation gesteuert werden. Liegt keine Ablagespezifikation vor, wird die aktuelle als Voreinstellung gewählt. Die Ablagespezifikation des REPL-Befehls dominiert über die Ablagesteuerung durch den STARR-Befehl und über Ablagespezifikationen. Treten im Replikationsblock Halbworte und Ganzworte auf, so wird dieser notfalls auf Ganzwortgrenze aufgefüllt. Die Spezifikationen U und G beim REPL-Befehl entsprechen einem vorausgehenden ASP 0/U bzw. ASP 0/G.

Im Replikationsblock dürfen die Pseudobefehle ASP, DSP, ABLAGE, AEND nicht vorkommen; Replikationen dürfen nicht geschachtelt werden. Literale sind verboten.

14.2. REND-Befehl

```
⟨REND-Befehl⟩   ::=  REND
```

Der REND-Befehl beendet den Wirkungsbereich eines vorhergehenden REPL-Befehls.

Beispiele

```
LI=     REPL    V (16),
        1,
        'FFFFFF'/H,
        REND,           :IN DIESEM FALL WIRD DER BLOCK UM EIN
                         LEERES HW VERLAENGERT, SO DASS 32
                         GW BELEGT WERDEN;
        STARR   V,
DP=     ''*021'',
        REPL    (20),
        'AFAFAFAFAFAF'/3,
        REND,
        STEND,
        M       TYP,
        S       1R,
        S       A,
        S       B,
        REPL    B (7),
        S       A,
        REND,
        S       B,
```

DIE TAS-MAKROSPRACHE

1.	Einleitung	1
1.1.	Elemente der Makrosprache	1
1.2.	Steuerung der Interpretation eines Makros	2
1.3.	Anwendung der Makrosprache	3
2.	Makrovariable	4
2.1.	Geltungsbereich der Makrovariablen	4
2.2.	Globale Makrovariable	5
3.	Makrokonstanten	5
4.	Definition von Makros	6
4.1.	DEF-Befehl	6
4.2.	Makros	6
4.3.	DEND-Befehl	7
5.	Aufruf von Makros	7
6.	Aufschlüsselung von Makrovariablenlisten	9
6.1.	FORM-Befehl	9
7.	Versionen	12
7.1.	VERS-Befehl	13
7.2.	SONST-Befehl	15
7.3.	VEND-Befehl	15
7.4.	Beispiele	15
8.	Wiederholungen	17
8.1.	WIED-Befehl	17
8.2.	Wiederholungstext	18
8.3.	WEND-Befehl	19
9.	Implizite globale Makrovariable	21
9.1.	NUMMER*	21
9.2.	DATUM*	22
9.3.	GENV*	22
9.4.	VERSION*	22

DIE TAS-MAKROSPRACHE

1. Einleitung

D

Die TAS-Makrosprache ist eine Erweiterung der TAS-Sprache. Sie enthält zusätzliche Pseudobefehle, die es erlauben, beliebige Folgen von Befehlen, Konstanten und anderen Informationseinheiten als sogenannte Makros zu definieren. Nach der Definition können diese Folgen durch sogenannte Makroaufrufe an beliebigen Stellen des Quellenprogramms eingesetzt werden.
Die einzelnen Pseudobefehle der Makrosprache ermöglichen es, Makros zu vereinbaren, aufzurufen und gegebenenfalls das Einsetzen der Makros zu steuern, um die generierte Folge dem Zweck des Aufrufs anzupassen.

1.1. Elemente der Makrosprache

Ein Makro wird nur einmal durch eine sogenannte Makrodefinition vereinbart und mit einem Namen versehen und kann dann unter diesem Namen beliebig oft durch Makroaufrufe aufgerufen und in das Quellenprogramm eingesetzt werden. Ein Makroaufruf ist also eine Anweisung an den Assembler, ein bestimmtes, vorher bereitgestelltes Makro zu interpretieren (interpretieren ist in diesem Fall gleichbedeutend mit einsetzen).

Damit ein Makro beim Einsetzen noch verändert werden kann, dürfen die einzelnen Informationseinheiten, aus denen ein Makro besteht, sogenannte Makrovariable enthalten, denen entweder beim Makroaufruf oder während der Interpretation des Makros aktuelle Werte, die sogenannten Makrokonstanten, zugeordnet werden. Makrovariable dürfen an beliebigen Stellen von Informationseinheiten stehen und werden unmittelbar vor der Interpretation der jeweiligen Informationseinheit durch ihren aktuellen Wert ersetzt.
Durch einzelne im Makro enthaltene Pseudobefehle kann auch direkt auf die aktuellen Werte von Makrovariablen Bezug genommen werden.

Ein bereits in einer Makrodefinition verwendeter Makroname kann bei weiteren Definitionen wieder verwendet werden, wodurch das alte Makro dieses Namens praktisch gelöscht wird.
Makronamen dürfen nicht mit den Namen der Befehle aus der Befehlsliste oder mit Namen von Pseudobefehlen identisch sein, da ein Makroaufruf nicht als solcher erkannt wird.

*) Die Verwendung von Makrovariablen in Pseudobefehlen ist nicht in allen Fällen möglich; bei ABLAGE- und beim ZONE-Befehl müssen die Zonen-Nummern explizit (und nicht über Makrovariable) angegeben werden.

1.2. Steuerung der Interpretation eines Makros

Beim Makroaufruf wird, falls vorhanden, der Adressenteil des Aufrufs einer bestimmten Makrovariablen als aktueller Wert zugeordnet. Über den Namen dieser Variablen kann im Inneren des Makros auf den Adressenteil des aktuellen Aufrufs Bezug genommen werden und damit (bei entsprechendem Aufbau des Makros) die Interpretation des Makros dem Zweck des Aufrufs entsprechend gesteuert werden (mit anderen Worten: der aufgrund eines bestimmten Makroaufrufs eingesetzte Text kann dem Adressenteil des Aufrufs entsprechend variiert werden).

Als Hilfsmittel, die einen Bezug auf den Adressenteil des aktuellen Aufrufs erlauben, und damit eine Variation der eingesetzten Folge ermöglichen, stehen neben den Makrovariablen die VERS-, WIED- und FORM-Befehle für die Verwendung in Makros zur Verfügung. Diese Pseudobefehle sind Anweisungen an den Assembler und werden während der Interpretation des Makros ausgeführt.

Ein VERS-Befehl ist die Anweisung an den Assembler, eine bestimmte Folge von Informationseinheiten, den Versionstext nur dann zu interpretieren, wenn die aktuellen Werte von bestimmten Makrovariablen bestimmten Bedingungen genügen.

Ein WIED-Befehl ist die Anweisung, eine bestimmte Folge von Informationseinheiten mehrfach zu interpretieren. Gleichzeitig vereinbart der WIED-Befehl eine Makrovariable, die bei jeder Interpretation der Folge einen neuen aktuellen Wert erhält, wodurch die Folge (ähnlich wie das Makro selbst) bei jeder Wiederholung variiert werden kann.

Der FORM-Befehl dient zur Aufschlüsselung von Makrokonstantenlisten, indem er in entsprechender Strukturierung Makrovariablennamen auflistet, denen die Elemente der Makrokonstantenliste zugeordnet werden. Insbesondere kann, falls der Adressenteil eines Makros eine Liste ist, durch einen im Makro enthaltenen FORM-Befehl diese Liste aufgeschlüsselt werden.

Alle Pseudobefehle der Makrosprache (wie Makroaufrufe, VERS-Befehle usw.) dürfen an beliebigen Stellen von Quellenprogrammen sowohl innerhalb als auch außerhalb von Makros verwendet werden. Es muß nur sichergestellt werden, daß die Makros und Makrovariablen, auf die diese Befehle Bezug nehmen, vor der Interpretation dieser Befehle vereinbart bzw. mit aktuellen Werten versehen sind.

1. 3. Anwendung der Makrosprache

Ganz allgemein erleichtern die Elemente der Makrosprache das Schreiben, die Korrektur und das Ändern von Quellenprogrammen.

Makroaufrufe können zur Erweiterung des Befehlsumfangs verwendet werden, sowie als Anweisung verstanden werden, häufig vorbereitete Programmteile einzusetzen.

Dem Benutzer eines ausgetesteten Makros ist es im allgemeinen gleichgültig, wie die aufgrund des Makroaufrufs generierte Befehlsfolge aussieht, wenn er weiß, welche genau definierte Leistung diese Befehlsfolge erbringt (siehe Kapitel G und H). Der Makroaufruf steht stellvertretend für diese Leistung und kann daher wie ein zusätzlicher Befehl oder Pseudobefehl verwendet werden (durch Anwendung des DRUCK-Befehls kann der Programmierer wählen, ob der aufgrund des Makroaufrufs generierte Quellentext im Übersetzerprotokoll erscheinen soll oder nicht; im letzteren Fall unterscheidet sich der Makroaufruf hinsichtlich der Protokollierung nicht von einem normalen Befehl). Mit Hilfe der Makrosprache können zur Ergänzung des Befehlsumfanges neue Befehle mit genau definierten Leistungen eingefügt werden. Der Benutzer dieser Befehle braucht dabei die Makrosprache nicht zu kennen; die Kenntnis der Leistungen der neuen Befehle genügt.

Diese Möglichkeit, einzelnen Befehlen bestimmte Leistungen zuzuordnen, kann auch ausgenutzt werden, um die Kompatibilität von Rechenanlagen, deren Ausstattung und Befehlsumfang nicht wesentlich verschieden ist, herzustellen: Aus einem Quellenprogramm können von ein und demselben Assembler Maschinenprogramme, die auf verschiedenen Anlagen laufen können, erzeugt werden. Bei den verschiedenen Übersetzungsvorgängen müssen dem Assembler nur verschiedene Sätze von Makrodefinitionen zur Verfügung gestellt werden, die gleiche Leistungen den Gegebenheiten der verschiedenen Anlagen entsprechend verschieden realisieren.

2. Makrovariable

⟨Makrovariable⟩	::=	+(⟨Makrovariablenname⟩)
⟨Makrovariablenname⟩	::=	⟨Name⟩

Eine Makrovariable ist ein Name, dem die Zeichenfolge + ((nicht durch Blank getrennt !) vorausgeht und dem eine schließende Klammer folgt. Durch bestimmte Pseudobefehle (Makroaufrufe, FORM-Befehl, WIED-Befehl) kann einer Makrovariablen eine sogenannte Makrokonstante als aktueller Wert zugeordnet werden. Makrovariable dürfen an beliebigen Stellen von TAS-Informationseinheiten stehen und werden unmittelbar vor der Interpretation der Informationseinheit durch ihren aktuellen Wert ersetzt.

Beispiele

Im folgenden Text kommen die Variablen namens A und S5 vor

```
L+(A)=  B       +(S5),
```

Angenommen diese Variablen haben die Werte 21 und C10 so entsteht bei der Interpretation die Folge:

```
L21=  B    C10,
```

2.1. Geltungsbereich der Makrovariablen

Der Geltungsbereich einer bestimmten Makrovariablen erstreckt sich immer auf genau auf ein bestimmtes Makro. Eine Variable ist also immer nur während der Interpretation eines Makros vereinbart.
Unterbricht der Assembler die Interpretation eines Makros, weil im Makro ein weiterer Makroaufruf enthalten ist, so ist auch die Gültigkeit aller im äußeren Makro enthaltenen Variablen für die Dauer der Interpretation des inneren Makros unterbrochen. Erst nach der Interpretation des inneren Makros sind wieder die Variablen des äußeren Makros gültig. Die einzige Ausnahme sind die globalen Makrovariablen.

2.2. Globale Makrovariable

Globale Makrovariable sind während der gesamten Dauer der Übersetzung eines Quellenprogramms vereinbart. Sie werden teils vom Assembler selbst vereinbart (siehe D.9), können aber auch im Zusammenhang mit dem WIED-Befehl definiert werden.
Außerdem sind alle über den FORM-Befehl aus dem globalen Namen VERSION* hervorgehenden Namen global (siehe D.6 und D.9.4).

3. Makrokonstanten

⟨Makrokonstante⟩	::=	{ ⟨Zeichenfolge, die die Abgrenzungszeichen ,|--|(|)|:|; sowie die Zeichen ␣|' nicht enthalten darf und nicht mit den Zeichen =|/|. beginnen darf⟩ ⟨selbstdefinierender Ausdruck⟩ ⟨Makrokonstantenliste⟩ }
⟨selbstdefinierender Ausdruck⟩		siehe Kapitel B.4
⟨Makrokonstantenliste⟩	::=	([[⟨Name⟩=] ⟨Makrokonstante⟩])

Makrokonstanten sind Zeichenfolgen, die durch bestimmte Pseudobefehle der Makrosprache einzelnen Makrovariablen als aktuelle Werte zugeordnet werden. Als Makrokonstanten können im einfachsten Fall beliebige Zeichenfolgen verwendet werden, die jedoch bestimmte, in der TAS-Sprache als Steuerzeichen verwendete Zeichenkombinationen nicht enthalten dürfen. Außerdem können als Makrokonstanten selbstdefinierende Ausdrücke und Listen von Makrokonstanten verwendet werden. Diese Listen enthalten einzelne Makrokonstanten, die eventuell mit einzelnen Namen versehen sein können.
Die Makrokonstanten werden vom Assembler als Zeichenstrings verwaltet; insbesondere werden die selbstdefinierenden Ausdrücke nicht umgewandelt (siehe auch D.5 und D.7).

Beispiele

```
315
H2S
''A,B,C''
(A=(ABC,3),B=2,C=((3,5),(6,7)))
```

4. Definition von Makros

```
⟨Makrodefinition⟩   ::=  ⟨DEF-Befehl⟩ ⟨Makro⟩ ⟨DEND-Befehl⟩
⟨DEF-Befehl⟩        ::=  DEF ␣ ⟨Makroname⟩
⟨Makroname⟩         ::=  ⟨Name⟩
⟨DEND-Befehl⟩       ::=  DEND
⟨Makro⟩             ::=  ⟨Informationseinheit⟩^n
```

Eine Makrodefinition vereinbart eine bestimmte Folge von Informationseinheiten, ein sogenanntes Makro. Eine Makrodefinition besteht aus einem DEF-Befehl, dem Makro und einem DEND-Befehl.

4.1. DEF-Befehl

Durch den DEF-Befehl wird das Makro eingeleitet. Im Adressenteil dieses Befehls ist der Name des Makros angegeben. Unter diesem Namen kann das Makro nach der Definition beliebig oft im Quellenprogramm aufgerufen werden. Der DEF-Befehl ist also eine Anweisung an den Assembler, das folgende Makro unter dem angegebenen Namen aufzubewahren.

4.2. Makros

Ein Makro ist eine beliebige Folge von Informationseinheiten. Es darf selbst wieder Makroaufrufe sowie Makrodefinitionen, Versionskomplexe und Wiederholungen enthalten (jeweils komplett). Bei der Definition wird das Makro nicht interpretiert, sondern unverändert aufbewahrt und erst bei jedem einzelnen Anruf wie eine normale Folge von Informationseinheiten interpretiert. Bei der Definition untersucht der Assembler nur die DEF- und DEND-Befehle, um das Ende des Makros feststellen zu können. Durch im Makro enthaltene Pseudobefehle kann auf den Adressenteil des Makroaufrufs Bezug genommen und damit die Interpretation des Makros durch den Assembler entsprechend dem Adressenteil des aktuellen Aufrufs variiert werden.

4.3. DEND-Befehl

Der DEND-Befehl beendet die Makrodefinition.

Beispiele

```
DEF     EMIL,

        BA      10,
        C       ANF,

DEND,
```

Nach dieser Definition kann die Folge BA 10, C ANF, mit dem Aufruf EMIL an beliebiger Stelle des Quellenprogramms eingesetzt werden

5. Aufruf von Makros

⟨Makroaufruf⟩ ::= [⟨Benennungsteil⟩] ⟨Makroname⟩ [⟨Makrokonstante⟩]

Ein Makroaufruf ist ein spezieller Pseudobefehl, dessen Befehlscode ein bestimmter Makroname ist. Dieser Pseudobefehl ist eine Anweisung an den Assembler, als nächstes das durch den Befehlscode angegebene Makro zu interpretieren. Ein Makroaufruf kann auch einen Adressenteil enthalten. In diesem Fall vereinbart der Assembler für die Dauer der Interpretation des Makros eine Makrovariable, die den Namen des Makros trägt und ordnet dieser Makrovariablen den Adressenteil des Makroaufrufs als aktuellen Wert zu. Auf diese Makrovariable kann dann im Inneren des Makros durch Pseudobefehle (FORM-Befehl, VERS-Befehl) Bezug genommen werden. Außerdem wird diese Makrovariable (wie jede andere Makrovariable) überall dort, wo sie im Inneren des Makros vorkommt, durch ihren aktuellen Wert (in diesem Fall den Adressenteil ersetzt). Der Adressenteil des Makroaufrufs kann auch eine Liste von einzelnen Werten sein, die durch einen im Makro selbst enthaltenen speziellen Pseudobefehl (den FORM-Befehl; siehe nächsten Abschnitt) anderen Makrovariablen als aktuelle Werte zugeordnet werden können. Durch die Angabe des Adressenteils kann also die Interpretation des Makros beeinflußt werden, und damit der interpretierte Text dem Zweck des Aufrufs angepaßt werden.

Alle im Inneren eines Makros oder beim Aufruf des Makros vereinbarten Makrovariablen sind nur während der Interpretation des Makros gültig. Nach der Interpretation des Makros sind diese Variablen nicht mehr vereinbart.

Die Benennung eines Makroaufrufs wird an den Makrotext weitergereicht. Die maximale Verschachtelungstiefe von Makroaufrufen ist (inklusive des äußersten Aufrufs) 8.

Beispiel

Das Makro DORA ist folgendermaßen definiert:

```
DEF     DORA,

        BA      10,
        C       +(DORA),

DEND,
```

Auf Grund des Aufrufs

```
DORA    SP1,
```

wird der Text

```
BA      10,
C       SP1,
```

erzeugt

Beispiel

```
DEF     STR,

        +(STR),
        '+(STR)',

DEND,
```

Der Aufruf STR '123',
ergibt '123',
"123",

6. Aufschlüsselung von Makrovariablenlisten

6.1. FORM-Befehl


```
⟨FØRM-Befehl⟩          ::=  FØRM ␣ ⟨Variablen-Name⟩ ⟨Zuordnungsliste⟩
⟨Zuordnungsliste⟩      ::=  ({⟨Variablen-Name⟩ [=⟨Vorbesetzungswert⟩]}...)
⟨Vorbesetzungswert⟩    ::=  ⟨Makrokonstante⟩
```

Der FORM-Befehl ist eine Zurodnungsanweisung. Er bezieht sich mittels eines Variablen-Namens auf den aktuellen Wert einer Variablen. Der aktuelle Wert dieser Variablen muß eine Liste von einzelnen Werten sein. Die in dieser Werte-Liste zusammengefaßten Werte ordnet der FORM-Befehl einzelnen Variablen als aktuelle Werte zu. Diese Zuordnung ist durch eine im FORM-Befehl enthaltene Zuordnungsliste festgelegt. Die Zordnungsliste enthält einzelne Variablennamen, die eventuell mit einem Vorbesetzungswert versehen sein können.

Die Zuordnung geht in 2 Stufen vor sich:
Zunächst werden allen Variablen, für die in der Zuordnungsliste Vorbesetzungswerte enthalten sind, diese Werte als aktuelle Werte zugeordnet.
Anschließend werden die Werte der Werteliste der Reihe nach den einzelnen in der Zuordnungsliste angeführten Variablen als aktuelle Werte zugeordnet (wobei eventuell Vorbesetzungswerte überschrieben werden).
Die Zuordnungsliste schreibt also vor, in welcher Reihenfolge die einzelnen Werte in der Werteliste angeordnet sein müssen: Diese Reihenfolge muß genau der Reihenfolge der Variablennamen in der Zuordnungsliste antsprechen. (Insbesondere schreiben die in einem Makro enthaltenen FORM-Befehle, die sich auf den Adressenteil des Aufrufs beziehen, das Format dieses Adressenteils vor).

Es ist jedoch auch möglich, beim Anschreiben der Werteliste von dieser Reihenfolge abzuweichen. In diesem Fall müssen jedoch die Werte, die nicht in der vorgesehenen Reihenfolge angeschrieben sind, mit dem Namen derjenigen Variablen, der sie durch den FORM-Befehl zugeordnet werden sollen, versehen sein.

Beispiele

Das Makro ANTON wurde durch folgende Makrodefinition vereinbart:

```
DEF     ANTON,
        FORM    ANTON (A,B,C),
        FORM    C  (D,E),

        BZ      +(C),
        CW      A+(D),
        C       A+(E),
        BA      +(B),
        AA      +(A),

DEND,
```

Der erste FORM-Befehl ordnet den Variablen A, B und C die Werte der Liste ANTON als aktuelle Werte zu (und schreibt damit gleichzeitig die Reihenfolge der Werte in dieser Werteliste, die als Adressenteil des Aufrufs fungiert, vor).

Der zweite FORM-Befehl ordnet den Variablen D und E die Werte der Werteliste C als aktuelle Werte zu. Auf Grund des Aufrufs

```
ANTON   (1,2,(3,4)),
```

wird daher die Folge

```
BZ      (3,4),
CW      A3,
C       A4,
BA      2,
AA      1,
```

übersetzt

In einzelne Schritte zerlegt, sieht der Zuordnungsvorgang folgendermaßen aus: Mit dem Aufruf

ANTON (1, 2, (3, 4)),

erhält die Variable ANTON den aktuellen Wert (1, 2, (3, 4)).

```
+(ANTON) ::=      (1, 2, (3, 4)),
                   ↓  ↓   ↓
FORM      ANTON (A, B,  C),
```

Der erste FORM-Befehl schlüsselt die Werteliste ANTON auf:

```
+(A)      ::=   1
+(B)      ::=   2
+(C)      ::=   (3, 4)
                  ↓  ↓
FORM        C  (D, E)
```

Der zweite FORM-Befehl schlüsselt die Werteliste C auf:

```
+(D)      ::=   3
+(E)      ::=   4
```

Das Makro BERTA wurde durch folgende Makrodefinition vereinbart

```
DEF       BERTA,
          FORM BERTA   (A=30,B=40,C=60),

          B        +(A),
          BH       +(B),
          BD       +(C),

DEND,
```

Aus dem Aufruf

```
BERTA     (B=50),
```

wird folgende Folge generiert:

```
B         30,
BH        50,
BD        60,
```

7. Versionen

⟨Versionskomplex⟩ ::= ⟨VERS-Befehl⟩ [⟨Versionstext⟩]
[⟨SØNST-Befehl⟩ [⟨Versionstext⟩]]ⁿ
⟨VEND-Befehl⟩

⟨Versionstext⟩ ::= { ⟨Versionskomplex⟩ | ⟨einfache IE⟩ }^1-n

⟨einfache IE⟩ ::= ⟨Informationseinheit, ausgenommen die Pseudobefehle VERS, VEND, SØNST⟩

⟨VERS-Befehl⟩ ::= VERS ⟨Bedingungsliste⟩

⟨SØNST-Befehl⟩ ::= SØNST [⟨Bedingungsliste⟩]

⟨VEND-Befehl⟩ ::= VEND

⟨Bedingungsliste⟩ ::= ((⟨Bedingung 1. Art⟩ | ⟨Bedingung 2. Art⟩))

⟨Bedingung 1. Art⟩ ::= ⟨Makrovariablenname⟩=[/⟨Bedingungsoperator⟩/]
⟨Makrokonstante⟩

⟨Bedingung 2. Art⟩ ::= ⟨Makrovariablenname⟩

⟨Bedingungsoperator⟩ ::= { ⟨gleich⟩ | ⟨ungleich⟩
⟨linksbündig kleiner gleich⟩ | ⟨linksbündig kleiner⟩
⟨linksbündig größer gleich⟩ | ⟨linksbündig größer⟩
⟨rechtsbündig kleiner gleich⟩ | ⟨rechtsbündig kleiner⟩
⟨rechtsbündig größer gleich⟩ | ⟨rechtsbündig größer⟩ }

⟨gleich⟩ ::= GL

⟨ungleich⟩ ::= UG

⟨linksbündig kleiner gleich⟩ ::= LKG

⟨linksbündig kleiner⟩ ::= LKL

⟨linksbündig größer gleich⟩ ::= LGG

⟨linksbündig größer⟩ ::= LGR

⟨rechtsbündig kleiner gleich⟩ ::= RKG

⟨rechtsbündig kleiner⟩ ::= RKL

⟨rechtsbündig größer gleich⟩ ::= RGG

⟨rechtsbündig größer⟩ ::= RGR

Ein Versionskomplex besteht aus ein oder mehreren Versionstexten, von denen der erste durch den VERS-Befehl, die weiteren durch einen SONST-Befehl eingeleitet werden; ein Versionstext wird durch den folgenden SONST-Befehl bzw. durch den VEND-Befehl abgeschlossen; dieser beendet auch den gesamten Versionskomplex.
Ein Versionstext ist eine Folge von Informationseinheiten und kann auch Makrodefinitionen, Wiederholungen und Versionskomplexe enthalten (jeweils komplett). Ist keine der in der Bedingungsliste enthaltenen Bedingungen erfüllt, so wird der Versionstext überlesen. Wenn der Versionstext interpretiert wird, werden alle darin enthaltenen Pseudobefehle normal ausgeführt und im Versionstext enthaltene Makrovariable durch ihre aktuellen Werte ersetzt. Insbesondere werden in einem Versionstext enthaltene weitere Versionskomplexe normal interpretiert (damit lassen sich ET-Verknüpfungen von Bedingungen durch Verschachtelung von Versionskomplexen und VEL-Verknüpfungen durch Zusammenfassung von Bedingungen in Bedingungslisten realisieren). Die maximale Verschachtelungstiefe beträgt 32.

7.1. VERS-Befehl

Der VERS-Befehl leitet den Versionstext ein. Er ist eine Anweisung an den Assembler, den folgenden Text nur zu übersetzen, wenn mindestens eine von mehreren im VERS-Befehl angegebenen Bedingungen erfüllt ist. Diese Bedingungen sind in einer Bedingungsliste zusammengefaßt. Diese Liste bildet den Adressenteil des VERS-Befehls. Es gibt 2 Arten von Bedingungen: Eine Bedingung 1. Art wird durch einen Variablennamen angegeben, dem eine Makrokonstante, ggf. mit Bedingungsoperator zugeordnet ist. Sie ist erfüllt (falls kein Bedingungsoperator vorliegt), wenn der aktuelle Wert der Variablen mit der Makrokonstanten übereinstimmt. Die Bedingungsoperatoren vergleichen die aktuelle Makrokonstante mit dem in der Bedingungsliste des die Version steuernden Befehls angegebenen Wert. Diese Prüfung kann man auf Gleichheit oder Ungleichheit oder auf die Bedingung "kleiner (gleich)" oder "größer (gleich)" von entweder linksbündig oder rechtsbündig gedachten Zeichenfolgen erfolgen lassen.

Wirkungsweise der Bedingungsoperatoren

Der Wert der Makrovariablen und die auf den Operator folgende Makrokonstante werden wie Oktadenstrings betrachtet. (Keine Konvertierung bei Zahlen!)

- Linksbündiger Vergleich: Haben die Strings ungleiche Länge, so denkt man sich den kürzeren String mit NUL-Oktaden verlängert (nicht Blanks!) Die erste Position von links her, auf der sich die beiden Strings unterscheiden, entscheidet den Vergleich; dabei ergibt sich die Anordnung der Zeichen aus dem Zentralcode.
  Beispiel:
  Hat die Makrovariable S den Wert ANNA, so ist die Bedingung
  S = /LGR/ ANDREAS erfüllt. (Die Werte unterscheiden sich zuerst im dritten Zeichen, und es ist N > D). Gibt es keine Position mit unterschiedlichen Zeichen, so sind die Strings gleich.

- Rechtsbündiger Vergleich: Haben die Strings ungleiche Länge, so ist der längere String rechtsbündig größer. Bei gleicher Länge stimmt der rechtsbündige mit dem linksbündigen Vergleich überein.
  Enthalten die Strings nur Ziffern, so ist der rechtsbündige Vergleich der normale Zahlenvergleich. (Achtung bei führenden Nullen - siehe Beispiele).

Beispiele

Die folgenden Vergleiche von Makrokonstanten sind wahr:

```
        17/RGR/8
        17/RGR/08
        17/RKL/008      (!)
        17/LKL/8        (Linksvergleich bei Zahlen ungeeignet)
MEYER   /LGR/LEHMANN
MEYER   /RKL/LEHMANN
         A/LGR/10
         A/RKL/10
         B/LKL/B1
       A*B/LKL/AB
        A2/LGR/A1
        A2/RGR/A1
```

Eine Bedingung 2. Art besteht nur aus einem Variablennamen. Sie ist bereits erfüllt, wenn der Variablen überhaupt ein beliebiger Wert zugeordnet ist.

7.2. SONST-Befehl

Der SONST-Befehl dient dazu, einen Versionskomplex in Alternativen aufzuteilen, die von jeweils anderen Bedingungen oder von überhaupt keiner Bedingung mehr abhängen. Diese Alternativen reichen jeweils von VERS bis SONST, von SONST bis SONST oder von SONST bis VEND. Die erste Alternative mit erfüllter Bedingung wird vom Assembler ausgewählt. Falls vorher noch keine Bedingung erfüllt war, wird die Alternative eines SONST-Befehls ohne Bedingung immer ausgewählt.

7.3. VEND-Befehl

Der VEND-Befehl schließt einen Versionstext ab.

7.4. Beispiele

```
VERS    (A=3,B=4),

        VERS    (C=5,D=7),
                B    (30),
        VEND,

VEND,
```

Der Text B (30) wird nur übersetzt, wenn die Bedingung
((A=3)∨(B=4))∧((C=5)∨(D=7)) erfüllt ist

Das Makro BERTA ist folgendermaßen definiert:

```
DEF     BERTA,
        FORM    BERTA  (VON,BIS,VZ=UNDEF),

        VERS    (VZ=UNDEF),
             B     +(VON),
        VEND,

        VERS    (VZ=N),
             BN    +(VON),
        VEND,

        C        +(BIS),

DEND,
```

Aus

```
BERTA    ((20),SP4,N),
```

wird:

```
BN       (20),
C        SP4,
```

Aus

```
BERTA    ((20),SP3),
```

wird:

```
B        (20),
C        SP3,
```

Das Makro MAKRO ist folgendermaßen definiert:

```
DEF      MAKRO,
FORM     MAKRO   (A,B),

         VERS    (A=0),
              BA      0,

         SONST,
              BA      1,

         VERS    (B),
              ZTR     3A,

         SONST,
              ZTR     2A,

         VEND,
         VEND,
DEND,
```

Aufgrund des Aufrufs

```
MAKRO    (1),        MAKRO    (0),        MAKRO    (1,3),
```

wird der Text

```
BA       1,          BA       0,          BA       1,
ZTR      2A,         ZTR      2A,         ZTR      3A,
```

interpretiert.

```
DEF     XYZ,
        VERS    (XYZ='10'),
        1,
        SONST   (XYZ='16'),
        2,
        SONST,
        3,
        VEND,
DEND,
```

Der Aufruf XYZ '16' ergibt 2,
XYZ '10' ergibt 1,
XYZ "*016" ergibt 3,

8. Wiederholungen

⟨Wiederholung⟩	::=	⟨WIED-Befehl⟩ ⟨Wiederholungstext⟩ ⟨WEND-Befehl⟩
⟨WIED-Befehl⟩	::=	⟨Makrovariablenname⟩[.]=WIED ⟨Werteliste⟩
⟨Werteliste⟩	::=	([⟨Makrokonstante⟩])
⟨Wiederholungstext⟩	::=	⟨Informationseinheit⟩^{n}
⟨WEND-Befehl⟩	::=	WEND

8.1. WIED-Befehl

Der WIED-Befehl ist die Anweisung an den Assembler, eine bestimmte Folge von Informationseinheiten, den WIED-Text, mehrfach zu interpretieren.
Der WIED-Text wird durch den WEND-Befehl abgeschlossen.
Die Anzahl der Wiederholungen ist direkt abhängig von der Anzahl der Werte in der Werteliste (WIED-Adressenteil).
Der WIED-Befehl vereinbart außerdem eine Makrovariable, die bei der ersten Interpretation den Wert der 1. Makrokonstanten bei der 2. Interpretation der Folge den Wert der 2. Makrokonstanten der Werteliste erhält, usw.
Durch Aufruf dieser Makrovariablen (WIED-Benennungsteil) im Wiederholungstext ist es möglich, die Folge bei jeder Wiederholung zu variieren. Nach Ablauf der Wiederholungen behält die Makrovariable den letzten Wert der Werteliste.

(Steht der WIED-Befehl außerhalb einer Makrodefinition, hat sein Name globale Bedeutung. Innerhalb einer Definition bleibt die WIED-Makrovariable lokal auf das aktuelle Makro beschränkt, es sei denn, die Benennung wäre mit einem Globalpunkt versehen).

8.2. Wiederholungstext

Der Wiederholungstext ist eine beliebige Folge von Informationseinheiten und darf selbst Wiederholungen und Versionskomplexe sowie Makrodefinitionen enthalten. Diese müssen komplett im Wiederholungstext enthalten sein. (Ein Wiederholungstext darf daher z. B. keinen einzelnen VERS-Befehl ohne den entsprechenden VEND-Befehl enthalten, da zu einem vollständigen Versionskomplex ein VERS- und ein VEND-Befehl gehören). Selbstverständlich darf im Inneren eines Wiederholungstextes auf alle Makrovariablen, denen ein aktueller Wert zugeordnet ist, mittels der dafür geeigneten Befehle (z. B. FORM-Befehl, VERS-Befehl) Bezug genommen werden. Bei der Interpretation des Wiederholungstextes werden die im Wiederholungstext enthaltenen Variablen durch ihre aktuellen Werte ersetzt.
Folgen in der Werteliste des WIED-Befehls zwei oder mehrere Kommata aufeinander; so wird die Angabe dieses leeren Wertes folgendermaßen interpretiert:

a) Die Makrovariable behält den alten Wert

b) Die Wiederholungsschleife wird mit diesem alten Wert nochmals durchgeführt.

8.3. WEND-Befehl

Der WEND-Befehl schließt einen Wiederholungstext ab.

Beispiele

Aus

```
A=      WIED   ((3,5),(7,8)),
               FORM  A  (B,C),

               B     +(B),
               C     +(C),

       WEND,
```

wird:

```
B          3,
C          5,
B          7,
C          8,
```

Aus

```
A=      WIED   (1,2),
B=      WIED   (3,4),

            B     +(A)+(B),

       WEND,
       WEND,
```

wird:

```
B          13,
B          14,
B          23,
B          24,
```

Das Makro TRANS ist folgendermaßen definiert:

```
DEF     TRANS,
        A=   WIED +(TRANS),

                C    +(A),

             WEND,
DEND,
```

Auf Grund des Aufrufs

```
TRANS     (SP1,SP2,SP3),
```

wird daher der Text:

```
C           SP1
C           SP2
C           SP3
```

generiert. Man beachte, daß der Befehl A=WIED+(TRANS) erst ausgeführt wird, nachdem die in diesem Befehl enthaltene Makrovariable durch ihren auf Grund des Makroaufrufs zugeordneten aktuellen Wert (SP1, SP2, SP3) ersetzt wurde.

```
W=        WIED (123,,,456,,789,,,,),
              +(W),
          WEND,
```

Ergebnis:

```
123
123
123
456
456
789
789
789
789
```

9. Implizite globale Makrovariable

9.1. NUMMER*

Diese globale Makrovariable kann überall in einem Quellenprogramm wie eine gewöhnliche Makrovariable aufgerufen werden.

Als Wert wird NUMMER* die vom Assembler mitgezählte Nummer des aktuellen Makroaufrufs zugewiesen. Bei geschachtelten Makroaufrufen werden die aktuellen Nummern in einem aufgebauten Nummernkeller reserviert und entsprechend der Verschachtelung wieder entnommen.
Es sind bis zu 8 Stufen möglich. Der Initialwert ist 0.

Beispiel

Es seien folgende Makros definiert:

```
                        DEF     PLUSODERMINUS,
                        SXI     MINUS+(NUMMER*),
                        A       (+(PLUSODERMINUS)),
                        S       ENDE+(NUMMER*),
MINUS+(NUMMER*)=        N       0,
                        SB      (+(PLUSODERMINUS)),
ENDE+(NUMMER*)=         N       0,
                        DEND,

                        DEF     INNENMAKRO,
KONSTANTE+(NUMMER*)=    +(NUMMER*),
VARIABLE+(NUMMER*)=     +(NUMMER*)/V,
                        DEND,

                        DEF     AUSSENMAKRO,
KONSTANTE+(NUMMER*)=    +(NUMMER*),
                        INNENMAKRO,
VARIABLE+(NUMMER*)=     +(NUMMER*)/V,
                        DEND,
```

Erste Makroaufrufe und Makroexpansionen:

```
                        AUSSENMAKRO,

KONSTANTE1=             1,

                        INNENMAKRO,
KONSTANTE2=             2,
VARIABLE2=              2/V,
VARIABLE1=              1/V,


                        PLUSODERMINUS   13,

                        SXI     MINUS3,
                        A       (13),
                        S       ENDE3,
MINUS3=                 N       0,
                        SB      (13),
ENDE3=                  N       0,
```

9.2. DATUM*

Dieser Makrovariablen wird zu Beginn der Übersetzung das Tagesdatum in der Form tt.mm.jj als Wert zugewiesen.

9.3. GENV*

Dieser Makrovariablen wird zu Beginn der Übersetzung, falls die Quelle in einer Datei liegt, deren Generations-Versionsnummer in der Form ggggvv als Wert zugewiesen, bei Fremdstring dagegen der Wert 000000.

9.4. VERSION*

In vielen Fällen gibt es mehrere Versionen eines bestimmten Quellenprogramms, die sich nur in wesentlichen Einzelheiten unterscheiden. (Beispiele solcher Versionen sind z.B.: Assembler mit verschiedenem Befehlsumfang; Dienstleistungsprogramme, die Sonderwünsche bestimmter Rechenzentren erfüllen; verschiedene Test-Versionen; Programme mit verschiedensprachigen Fehlermeldungen usw.).

Die TAS-Makrosprache erlaubt es, solche verschiedenen Versionen eines Programms durch ein einziges Quellenprogramm darzustellen. Dadurch kann (da Änderungen nur in einem einzigen Quellenprogramm durchgeführt werden müssen) der Maintenanceaufwand u.U. bedeutend herabgesetzt werden.

Die Erzeugung verschiedener Versionen geht folgendermaßen vor sich:
Im UEBERSETZE-Kommando (siehe Kapitel E) kann als Wert der Spezifikation VERSION eine Makrokonstante angegeben werden, die die gewünschte Version kennzeichnet. Diese Makrokonstante wird vor Beginn der Übersetzung des Quellenprogramms einer globalen Makrovariablen mit dem Namen VERSION* als aktueller Wert zugeordnet. Diese Variable kann im gesamten Quellenprogramm wie eine normale Makrovariable verwendet werden, um die Interpretation des Quellenprogramms zu steuern.

So kann z.B. auf diese Variable mit Hilfe des VERS-Befehls Bezug genommen werden oder falls als Wert der Spezifikation VERSION eine Liste angegeben wird, kann diese Liste mit Hilfe des FORM-Befehls aufgeschlüsselt werden.

Außerdem darf diese Variable natürlich in beliebigen Informationseinheiten des TAS-Quellenprogramms auftreten. Die Steuerung der Interpretation eines TAS-Quellenprogramms verläuft also vollständig analog zur Steuerung der Interpretation eines Makros. In dieser Analogie entspricht das Quellenprogramm dem Makro, das UEBERSETZE-Kommando dem Makroaufruf und der Wert der Spezifikation VERSION dem Adressenteil des Makroaufrufs

Beispiele

Fall der einfachen Makrokonstanten UEBERSETZE-Kommando

```
◊UEBERSETZE,SPRACHE=TAS,VERSION=KN4,QUELLE=/
```

Aus den Versionsfolgen

```
VERS      (VERSION*=KN2),
          0,
SONST     (VERSION*=KN4),
          1,
SONST,
          2,
VEND,
```

wird die Informationseinheit

```
1,
```

in die Quelle eingesetzt.

Fall der Makrokonstantenliste UEBERSETZE-Kommando

```
◇UEBERSETZE,SPRACHE=TAS,VERSION=(KN4=JA),QUELLE=/
```

Aus dem folgenden Programmausschnitt mit Versionsfolgen

```
FORM    VERSION* (KN4,BO,M,HH,SEGM),

        VERS     (KN4),
           BA       4,

        SONST,
          BA        1,

        VEND,

          C3       VB30+7,
           XBA     VB30,
           SSR     30,

        VERS   (SEGM),
           B        VB,
           SFB      S&LOLA,
        VEND,

           S        L1,
```

werden diese Informationseinheiten in die zu übersetzende Quelle aufgenommen:

```
BA      4,
C3      VB30+7,
XBA     VB30,
SSR     30,
S       L1,
```

UEBERSETZE-KOMMANDO

1.	Quelle	1
2.	Sprache	1
3.	Numerierung	1
4.	Name des Montageobjekts	2
5.	Variante	2
6.	Protokoll	2
7.	Dynamische Kontrollen	2
8.	Trace	2
9.	Maintenance-Nummer	2
10.	Kontrollereignisse	3
11.	Transfer	3
12.	Version	3

Hier werden die Spezifikationen des Übersetze-Kommandos insoweit erklärt, als die für den TAS-Programmierer notwendigen Informationen über den Rahmen des Kommando-Handbuches hinausgehen.

1. Quelle

Keine weiteren Erklärungen.

2. Sprache

Die Angabe von TASE bedeutet, daß der eigentlichen Assemblierung ein Vorlauf vorausgeht, in dem die gesamte Quelle gelesen wird und die Ersetzungen ausgeführt werden. Zur Assemblierung wird dann diese neue Quelle noch einmal vom Hintergrund eingelesen. Wegen des Aufwandes lohnen sich also wenige Ersetzungen nicht.

Zur Angabe von TASR siehe die Beschreibung des Rahmenprogramms in Kapitel G.

3. Numerierung

Die Angabe -STD- bedeutet im Falle der Eingabe als Fremdstring, daß die einzelnen Informationseinheiten der Quelle in Zehnerschritten, beginnend bei 10, numeriert werden; im Falle der Eingabe aus einer Datei wird die vorhandene Satznummer übernommen. In diesem Fall und bei explizit gewünschter Quellzeilennumerierung erhalten alle Informationseinheiten der Zeile die gleiche Nummer. Informationseinheiten, die zu mehr als einer Quellzeile gehören, werden so numeriert, als ob sie vollständig zu der letzten dieser Zeilen gehörten.

Bei externer Numerierung erfolgt keine Prüfung auf Monotonie der Folge. Außerdem ist dann der Spezifikationswert R zur Spezifikation PROTOKOLL unwirksam.

Bei einer expliziten Nummerierung bzw. bei einer Übernahme der Zeilennummerierung aus einer Datei ist darauf zu achten, dass der Anfangswert < 400000 ist. Bei einem Anfangswert > 400000 läuft der TAS-Assembler auf einen arithmetischen Alarm.

4. Name des Montageobjekts

Zur Benennung des Montageobjekts im Falle -STD- siehe Beschreibung des SEGM-Befehls.

Mit einem UEBERSETZE-Kommando nur e i n Montageobjekt erzeugt werden.

5. Variante

Wird für ein Montageobjekt Dumpfähigkeit verlangt, so werden vom Assembler Alle Benennungen in den Schreibzonen für den TAS-Dumpoperator aufbewahrt. (Um den Dump zu gewährleisten, muß in der TAS-Quelle die Kontrollprozedur S&CC mittels EXTERN S&CC angeschlossen und mittels LR 1A, SFB S&CC+1 aufgerufen werden).

Im Falle VARIANTE = GS werden die Benennungen aus allen Zonen aufbewahrt; durch den KE-Befehl können Kontrollereignisse festgelegt werden (siehe Kapitel H. 4).

6. Protokoll

Das Übersetzungsprotokoll ist in Kapitel F beschrieben.

7. Dynamische Kontrollen

Entfällt bei TAS.

8. Trace

Im Falle "undefiniert" ignoriert der Assembler jeden Testbefehl in der Quelle und außerdem die auf ihn folgende Informationseinheit, falls es sich um einen Testbefehl mit Versorgung handelt. Die Testbefehle sind in Kapitel H beschrieben.

9. Maintenance-Nummer

Die Maintenance-Nummer wird in die entsprechende Montagecode-Deklaration umgesetzt.

10. Kontrollereignisse

Entfällt bei TAS (siehe aber E. 6).

11. Transfer

Das gesamte Montageobjekt wird als zuladbar erklärt (mit der angegebenen Vorrangnummer).

12. Version

Der Wert dieser Spezifikation ist eine Makrokonstante, die die Makrovariable VERSION* besetzt. Einschränkungen durch die Syntax der Kommandosprache sind zu beachten: In geklammerten Ausdrücken sind alle TAS-Zeichen erlaubt, sonst siehe Definition von Arbeitsstring.

Wenn die Konstante eine Liste ist, muß zunächst ein FORM-Befehl gegeben werden, bevor auf die Werte in dieser Liste zugegriffen werden kann.
Siehe dazu Beschreibung und Beispiele in Kapitel D. 9 sowie Kapitel D. 1. 2, letzter Absatz.

DAS TAS-PROTOKOLL

1.	Der Protokollrahmen	1
2.	Das Quelleprotokoll	2
2.1.	Unterdrückung des Quelleprotokolls	2
2.2.	Seitenvorschub und Kopfzeile	3
2.3.	Druckbreite	3
2.4.	Die Informationseinheiten	3
2.4.1.	Numerierung	3
2.4.2.	Objektcode (falls verlangt)	3
2.4.3.	Quelle	4
2.4.4.	Ganzwortliterale	5
3.	Abbruch der Übersetzung in Fehlerfällen	5
3.1.	Alarme (außer Ereignisalarmen)	5
3.2.	Nicht erbrachte Dienstleistungen	5
3.3.	Besondere Fälle	6

F

DAS TAS-PROTOKOLL

1. Der Protokollrahmen

Der Assembler meldet sich unmittelbar nach dem Start unter Angabe der Maintenanceversionsnummer.

Nach Abschluß der Protokollierung der Quelle werden folgende Angaben gemacht:

- Fehlermeldungen, die Kontaktnamen und Gebietsdeklarationen betreffen (falls Fehler vorliegen).
- Der Indexpegel, d.i. die höchste einem (symbolischen) Indexnamen zugewiesene Adresse, wird als Dezimalzahl ausgegeben.
- Falls durch das UEBERSETZE-Kommando das Protokoll unterdrückt wurde, erscheint (zur Kontrolle) die Anzahl der Informationseinheiten.
- Es werden die Seiten des Protokolls angezeigt, auf denen Fehlermeldungen zu finden sind (falls Fehler vorliegen); Bezugspunkt ist dabei die Seitennummer in der TAS-Kopfzeile, nicht die der Systemkopfzeile.
- Nach Ablage des Montageobjekts wird der in die Datei &MO eingetragene Montageobjektname ausgedruckt, gegebenenfalls mit der Meldung, daß ein gleichbenanntes älteres Montageobjekt gestrichen wurde. (Dieser Punkt entfällt, falls nur Syntaxprüfung verlangt wurde.)
- Falls gewünscht, wird nun das Adreßbuch ausgegeben, wobei zuerst die globalen Namen erscheinen, dann segmentweise unter Angabe des Segmentnamens (falls vorhanden) die lokalen Namen. Wurden zusätzlich Referenzen gewünscht, so erhält man pro Zeile ein Adreßbuchelement, gefolgt von der Liste der Referenzen, die sich auf die Numerierung der Protokollzeile beziehen. Die Referenzennummer 0 bedeutet, daß der Name bei einem EINGG-Befehl aufgeführt ist. Zur Ausgabe des Adreßbuchs mit Referenzen gehören abschließend die Merklichterreferenzen (einschließlich der Aufzählung aller BCL- und QBR-Befehle).
  Ohne Referenzen werden vier Adreßbuchelemente pro Zeile gedruckt (bei schmalem Druck nur zwei).

Format der Adreßbuchelemente (je 28 Druckspalten):

Spalte 1: Ablagebereich, dabei bedeutet
E Externer Name
I Indexname
G gleichgesetzter Name
(K, V, B, D wie üblich)

Spalten 2, 3: Zonennummer (hexadekadisch), entfällt bei I; bei E handelt es sich um eine Pseudozonennummer (für die Montage); bei G handelt es sich um eine Verweisnummer, unter der am Ende des Adreßbuchs der zugehörige Gleichsetzungstext ausgedruckt ist.

Spalten 4-9: Zonenrelative Adresse (hexadekadisch) entfällt bei G, ist bei E stets 0 und bei I eine "absolute" Adresse.

Spalten 13-24: Name (linksbündig); bei Namen von mehr als 12 Zeichen Länge werden nur die ersten zwölf Zeichen gedruckt.

Anmerkung: Das referenzlose Adreßbuch wird hauptsächlich zur Auswertung von Dumps und (unter Einbeziehung des Montageprotokolls) zur Herstellung von Korrekturen (in interner Form) benutzt.

Der Assembler verabschiedet sich mit der Angabe der verbrauchten Rechenzeit (Abwicklerzeit).

2. Das Quelleprotokoll

2.1. Unterdrückung des Quelleprotokolls

Die Protokollierung der Quelle kann über die Spezifikation PROTOKOLL des UEBERSETZE-Kommandos ganz oder mittels des Pseudobefehls DRUCK teilweise unterdrückt werden. Informationseinheiten, die Fehlermeldungen hervorrufen, werden in jedem Fall protokolliert. Bei abgeschaltetem Protokoll erscheint vor fehlerhaften Einheiten die letzte vorangegangene Überschrift.

2.2. Seitenvorschub und Kopfzeile

Vor der ersten Zeile des Quellprotokolls erfolgt ein Seitenvorschub. Pro Seite werden 59 Zeilen gedruckt; unabhängig davon bewirkt eine Überschrift das Vorrücken auf die nächste Seite.

Zu Beginn jeder Seite wird eine Kopfzeile gedruckt, die den Quellprogrammnamen (Benennung des ersten SEGM-Befehls), den aktuellen Segmentnamen (jeweils bis zu 12 Zeichen), die zuletzt erschienene Überschrift (bis zu 60 Zeichen, bei schmalem Protokoll bis zu 30 Zeichen) sowie die Seitennummer enthält.

2.3. Druckbreite

Es werden bis zu 131 Spalten bedruckt, bei schmalem Protokoll (PROT. =S oder KO) bis zu 69 Spalten.

2.4. Die Informationseinheiten

2.4.1. Numerierung

Bei expliziten Wünschen zur Numerierung erhält man eine Nummer pro Quellzeile (Karte, Datensatz), die dann i. a. für mehrere Informationseinheiten gültig ist; ansonsten werden die Informationseinheiten durchnumeriert.

Vom Assembler erzeugte Einheiten, wie eingeschobene Halbworte und Arbeitsspeicher werden durch einen Stern (*) markiert, ausgeführte Makros durch &.

2.4.2. Objektcode (falls verlangt)

- Die Zonennummer (erscheint nur bei Änderung der Ablagezone).
- Die zonenrelative Adresse (hexadekadisch) bezieht sich auf die zuletzt erschienene Zonennummer.
- Die Typenkennung; das Zeichen - bedeutet, daß der (Halbwort-) Wert keine eigene Typenkennung erhält (Halbworte auf ungerader Adresse).

- Die Werte:
  Bei Ganzworten wird das entstandene Bitmuster in Form von 12 Tetraden gedruckt.

  Bei Halbworten erscheint zunächst ein Translationsschlüssel, der folgende Werte annehmen kann:

0 Der Halbwortwert wird bei der Montage nicht verändert.

1 (bei Befehlen) Auf den Adreßteil (Bits 9 bis 24) wird die Anfangsadresse der angegebenen Zone addiert (modulo 2^{16}).

2 (bei Adreßkonstanten) Auf das Feld der Bits 3 bis 24 wird die Anfangsadresse der Zone addiert (modulo 2^{22}).

4 (bei Oktadenadreßkonstanten) Auf den Halbwortwert wird das Dreifache der Anfangsadresse der angegebenen Zone addiert (modulo 2^{24}).

Amerkung: Bei externen Namen wird mit der bei der Montage ermittelten Adresse translatiert.

Auf den Translationsschlüssel folgt die Zonennummer (der Zone, zu der der im Adreßteil benutzte Name gehört).

Beim Translationsschlüssel 0 erscheint auf der Position der Zonennummer eine Null.

Schließlich wird der (noch nicht translatierte) Halbwortwert in Form von 6 Tetraden ausgedruckt; bei Befehlen ist der Codeteil durch eine Leerstelle vom Adreßteil getrennt.

Anmerkung zum Objektcode: Mit Hilfe des Montageprotokolls, das die Anfangsadressen der montierten Zonen anzeigt, kann vom Objektcode im TAS-Protokoll auf das fertige Programm geschlossen werden.

2.4.3. Quelle

Die Benennungen sind gegenüber den Hauptteilen um 12 Spalten nach links herausgerückt.

Für den Hauptteil sind 22 Druckspalten reserviert; Konstanten werden rechtsbündig eingerückt; bei Befehlen wird der Befehlscode linksbündig eingerückt, der (oder die) Adreßteil (e) wird (werden) rechtsbündig eingerückt.

Anmerkung: Zu lange Namen oder Konstanten zerstören das gleichmäßige Druckbild.

Für Kommentare ist der Rest der Druckspalten vorgesehen. Selbständige Kommentare nehmen die Position des Hauptteils ein. Bei schmalem Protokoll beginnen Kommentare auf einer neuen Zeile.

Fehlermeldungen erscheinen unter der beanstandeten Einheit. Bei fehlerhaften Einheiten wird stets der Objektcode gedruckt.

Anmerkung: Die Fehlermeldungen sind zum Teil Warnungen, die den Programmlauf nicht notwendig beeinträchtigen.

2.4.4. Ganzwortliterale

Die Werte der Ganzwortliterale werden geschlossen hinter der letzten Informationseinheit ausgedruckt.

3. Abbruch der Übersetzung in Fehlerfällen

3.1. Alarme (außer Ereignisalarmen)

Der Alarmkeller wird in Standardform ausgedruckt.

Bei Abbruch vor Beginn der Quellprotokollierung (i.a. Fehlverhalten des Assemblers bei Syntaxfehlern) wird der Quelltext der zuletzt bearbeiteten Informationseinheit und das letzte vorausgegangene Label ausgegeben, um die Berichtigung der Quelle zu ermöglichen.

Der Variablenbereich des Assemblers wird gedumpt.

3.2. Nicht erbrachte Dienstleistungen

Es erscheint der Name des Dienstunterprogramms oder die Nummer des SSR, jeweils mit einem Fehlertext.

Anmerkung: Der Benutzer kann hier oft selbst Abhilfe schaffen (z.B. bei Überschreitung der Speicherberechtigung).

3.3. Besondere Fälle

In besonderen Fällen wie Listenüberläufe oder Erkennen "unmöglicher" Übersetzungszustände u. a. m. wird die Übersetzung mit dem Text "TAS-Abbruch" vorzeitig beendet. Hier gilt ebenfalls das unter F. 3. 1 gesagte.

Überschreitet der Anteil der fehlerhaften Einheiten einen gewissen Prozentsatz, so wird die Übersetzung während der Protokollierung abgebrochen.

DER STANDARDRAHMEN FÜR TAS-PROGRAMME

1.	Überblick	1
2.	Allgemeine Regeln für den Standardrahmen	2
3.	Das Makro R&RAHMEN	4
4.	Makros für die Eingabe	6
4.1.	Das Makro R&LIES	6
4.2.	Der Eingabepuffer-Zeiger	7
4.2.1.	Das Ma kro R&EPOS	8
4.2.2.	Das Makro R&ESKIP	8
4.3.	Makros zum Manipulieren von Eingabeinformation	9
4.3.1.	Das Makro R&EOKT	9
4.3.2.	Das Makro R&ETET	10
4.3.3.	Das Makro R&EGANZ	11
4.3.4.	Das Makro R&EBRUCH	11
4.3.5.	Das Mrko R&EGLEIT	12
4.3.6.	Das Makro R&WIED	12
5.	Makros für die Ausgabe	14
5.1.	Der Ausgabepuffer-Zeiger	14
5.1.1.	Das Makro R&APOS	14
5.1.2.	Das Makro R&ASKIP	15
5.2.	Makros zum Zusammensetzen von Ausgabeinformation	15
5.2.1.	Das Makro R&AOKT	15
5.2.2.	Das Makro R&ATET	16
5.2.3.	Das Makro R&AGANZ	16
5.2.4.	Das Makro R&ABRUCH	17
5.2.5.	Das Makro R&AGLEIT	17
5.3.	Das Makro R&DRUCKE	18
5.3.1.	Das Makro R&DRUCKETEXT	19
5.4.	Das Makro R&DRUCK	19
5.4.1.	Das Makro R&DRUCKTEXT	19
5.5.	Das Makro R&SEITE	20
5.6.	Das Makro R&VORSCHUB	20
5.7.	Das Makro R&DUMP	20
5.8.	Das Makro R&BINAERDUMP	21

6.	Makros zum Beenden der Programmausführung	21
6.1.	Das Makro R&FEHLER	21
6.2.	Das Makro R&ENDE	22
6.3.	Die Adresse R&ENDE	23
6.4.	Die Adresse R&SSRFEHLER	23
7.	Beispiel	24

DER STANDARDRAHMEN FÜR TAS-PROGRAMME

1. Überblick

Der "Standardrahmen" erleichtert das Schreiben von TAS-Programmen, indem er einige Standardvereinbarungen in Kraft setzt und Ein-Ausgabe-Routinen verfügbar macht. Im Kapitel G. 7 findet man ein Beispiel für ein gerahmtes TAS-Programm, das zur ersten Orientierung dienen kann.

Der Standardrahmen schafft auch einige der Voraussetzungen, die für einen TAS-Variablen-Dump im Fall eines Alarms oder SSR-Fehlers nötig sind. Zusätzlich sind durch den Benutzer nur noch folgende Bedingungen zu erfüllen:

- Im UEBERSETZE-Kommando muß VARIANTE = D stehen,
- Das STARTE-Kommando muß Angaben für DUMP enthalten, z. B. DUMP = T-ALLES.

Für den Anschluß des Überwachers sind nur noch nötig:

- Die Angabe TRACE=-STD- im UEBERSETZE-Kommando,
- Falls vom Übersetzer mehr (oder weniger) als maximal 30 Seiten ausgedruckt werden sollen, d. h. z. B. mehr als 1800 mal die Registerstände: eine Angabe nach UEBWS im STARTE-Kommando,
- Im TAS-Programm der Befehl TEST*EIN und weitere Test-Befehle, welche die zu überwachenden Befehle und Variablen angeben. Will man das gesamte Programm mit Ausnahme des Rahmens überwachen, so legt man an den Programmanfang die Befehle

  ANFANG. = NULL 0, TEST*EIN,
  TEST*FR, NULL(ANFANG/A, SCHLUSS/A, 'FFFFFF'/H),
  und unmittelbar vor den ENDE-Befehl
  SCHLUSS. = NULL 0,

Bei Auftreten von Fehlern veranlaßt der Rahmen u. a. den Ausdruck des Fehlerorts, und zwar als sedezimale absolute Adresse. Ihre Beziehung zur Programmniederschrift läßt sich erst herstellen, wenn auch Translationsgröße und Relativadressen bekannt sind; um sie zu erhalten, sind folgende Voraussetzungen zu erfüllen:

- Die Protokollierung durch den TAS-Übersetzer darf nicht ausgeschaltet sein (durch den Befehl DRUCK 0 wäre sie ausgeschaltet).

- Im UEBERSETZE-Kommando muß mindestens PROTOKOLL = O angegeben sein.

- Im MONTIERE-Kommando muß mindestens PROTOKOLL = A angegeben sein.

Der Standardrahmen besteht aus einigen Montageobjekten und Makrodefinitionen, die in der öffentlichen Datenbasis liegen und durch Angabe der Spezifikation SPRACHE = TASR des UEBERSETZE-Kommandos verwendbar werden.
Die Angabe TASR ist gleichwertig mit der Angabe TAS (nicht TASE) für ein TAS-Programm, vor das die Befehlsfolge

```
DRUCK 0
HDEF &OEFDB (R&RAHMEN),
R&RAHMEN,
DRUCK 2,
```

gelegt wurde. Falls im UEBERSETZE-Kommando für das gerahmte TAS-Programm die Spezifikation MO nicht vorkommt, erhält das Programm den (echten) Montageobjektnamen STDHP.

- Für Sonderfälle: Will man den Standardrahmen in mehreren Quellen benutzen, so darf man nur diejenige Quelle mit der Spezifikation SPRACHE = TASR übersetzen, die den Startpunkt enthält; die übrigen müssen mit SPRACHE = TAS (oder TASE) übersetzt werden und (spätestens) vor dem ersten Aufruf eines Makros des Standardrahmens die Pseudobefehle

  ```
  HDEF &OEFDB (R&RAHMEN),
  R&RAHMENFUERUNTPR,
  ```

  enthalten (die EXTERN-Befehle heranschaffen).

2. Allgemeine Regeln für den Standardrahmen

Alle vom Standardrahmen vereinbarten globalen und externen Namen, die auch im Benutzerprogramm gelten, beginnen mit den 2 Zeichen R&.

Die Argumente von Makros des Standardrahmens werden in Klammern eingeschlossen (d.h., von der in der TAS gegebenen Möglichkeit, Makros mit einem ungeklammerten Argument aufzurufen, wird nicht Gebrauch gemacht).

In den Syntaxformeln ist mit n stets eine natürliche Zahl gemeint.

Einige der Makros des Standardrahmens ändern den Inhalt der Rechenwerksregister und des Registers B, ohne daß in ihrer Beschreibung darauf hingewiesen wird. Merklichter, Unterprogramm-Ordnungszähler, Indexbasis und Indexzellen 0 bis 247 werden vom Rahmen nicht geändert. Die Indexzellen 248 bis 255 sind für die Rücksprungadressen von SU-Befehlen vorgesehen; der Rahmen verwendet hiervon nur eine SU-Unterprogrammstufe, die übrigen stehen dem Rahmenbenutzer zur Verfügung.

Sämtliche von den Makros des Standardrahmens erzeugten Befehle (außerhalb von Literalen) haben keine explizite Ablagespezifikation (werden also normalerweise in die implizite B0-Zone ablegt und können durch STARR-Befehle noch beeinflußt werden).

Bei Aufruf der meisten Makros (genauer: bei R&LIES, R&EOKT, R&ETET, R&EGANZ, R&EBRUCH, R&EGLEIT, R&DRUCKE, R&AOKT, R&ATET, R&AGANZ, R&ABRUCH, R&AGLEIT, R&FEHLER) darf kein BÜ- oder TK-Alarm anstehen (der beim nächsten Rechenwerksbefehl einen Alarm auslösen würde); gegebenenfalls wird der Fehler

BUE- ODER TK-ALARM BEIM EINTRITT IN RAHMEN

gemeldet und der Operatorlauf abgebrochen.

Bei Alarm, SSR-Fehler, einem vom Standardrahmen entdeckten Fehler und Ausführung einer vom Makro R&FEHLER (siehe Kapitel F.5.1) erzeugten Befehlsfolge geschieht folgendes:

- Gegebenenfalls wird der Überwacher ausgeschaltet.
- Fehlertext, der Art und Ort des Fehlers erkennen läßt, wird ausgedruckt.
- Gegebenenfalls werden gemäß der Spezifikation DUMP des STARTE-Kommandos Variable samt ihren Namen gedumpt (z.B. bei TK1 als echter Bruch). Zu diesen Variablen gehören normalerweise auch die folgenden des Standardrahmens: R&EPUFFER, R&EPUFFERPOS, R&APUFFERPOS, R&VORSCHUB, R&APUFFER, R&APUFFERVERLAENGERUNG, die insgesamt 226 Halbworte belegen, d.h. höchstens 2 Druckseiten liefern.

- Bei Software-Alarm (z. B. Überschreitung der im Abschnittskommando angegebenen Zeit- oder Druckseitenschranke) wird der Abschnitt abgebrochen, sonst, also u. a. bei SSR-Fehler oder Hardware-Alarm (z. B. Typenkennungs-Alarm, arithmetischer Alarm, Speicherschutzalarm, Überlauf des Registers U, ungültiger Befehlscode) wird der Operatorlauf beendet und zum nächsten Kommando übergegangen.

Anmerkung: Man beachte, daß TAS-Makro-Aufrufe keine Benennung haben dürfen. Auswege werden aus folgenden Beispielen deutlich

```
BEN = ASP 0/B, R&LIES,
BEN = NULL 0, R&LIES,
```

3. Das Makro R&RAHMEN

Überblick: Dieses Makro nimmt dem Benutzer des Standardrahmens die Arbeit ab, die START-, ALARM-, XBASIS-, UNTPR- und VORBES-Befehle zu schreiben, Speicherplatz für Indexregister zu reservieren und Alarm-, Eingabe- und Ausgabeprozeduren zu schreiben und anzuschließen.
Nach diesen Vorbereitungen sind dann die übrigen Makros des Standardrahmens aufrufbar.

Aufruf: Zweckmäßigerweise ruft man das Makro R&RAHMEN implizit mit Hilfe der Kommando-Spezifikation SPRACHE = TASR auf (siehe Kapitel G. 1).

Wirkung: Es wird ein Speicherbereich für 256 Indexzellen reserviert und ein XBASIS-Befehl mit der Anfangsadresse dieses Bereichs erzeugt.

Ein Befehl UNTPR 247, wird erzeugt.

Ein Befehl VORBES (3, 'FFFFFFFFFFFF'), wird erzeugt.

Als Eingabepuffer werden 80 Halbworte reserviert; sie sind mit 16 Bits adressierbar; das erste Halbwort hat den (globalen EINGG-) Namen R&EPUFFER. Außerdem wird der zur Eingabe eines evtl. vorhandenen DATEN-Fremdstring des STARTE-(oder RECHNE-) Kommandos erforderliche Anfangsaufruf durchgeführt.

Als Ausgabepuffer werden 116 Halbworte reserviert; sie sind mit 16 Bits adressierbar; das erste Halbwort hat den (globalen EINGG-) Namen R&APUFFER. Anfangs wird jedes Halbwort des Puffers mit NUL NUL SP belegt, als Papiervorschub (in dem Ganzwort R&VORSCHUB) "nächste Zeile" vorgesehen und der Ausgabepuffer-Zeiger (R&APUFFERPOS) auf das erste Halbwort des Puffers gerichtet.

Die für die Ein- und Ausgabe nötigen Montageobjekte (R&MO, S&GZF, S&KDEG, S&KEGD) werden angeschlossen.

Zur Behandlung von Alarmen und SSR-Fehlern wird ein ALARM-Befehl erzeugt, die Montageobjekte S&CC und S&SRF angeschlossen und die erforderlichen Anfangsaufrufe durchgeführt.

Die im STARTE-Kommando nach UEBWS möglicherweise angegebene Druckseitenschranke für den Überwacher wird mit Hilfe des Befehls TEST*SSATZ dem Überwacher mitgeteilt, andernfalls wird ihm UEBWS = 30 mitgeteilt (die evtl. vom Benutzer angegebenen weiteren TEST*SSATZ-Befehle bleiben unbeachtet).

Es wird ein START-Befehl erzeugt, der auf die gemäß den o.a. Regeln erzeugte Befehlsfolge springt. Diese endet mit einem Sprung auf das erste vom Benutzer in die implizite B0-Zone abgelegte Halbwort. Der Rahmen sorgt dafür, daß dieses Halbwort auf den Anfang einer Achtelseite kommt, so daß Überwacherprotokolle leichter lesbar sind.

Anmerkungen: Von der standardmäßigen Wahl für Unterprogramm-Ordnungszähler, Indexbasis und Alarmadresse kann man nachträglich leicht durch die entsprechenden TR 440-Befehle (speziell auch SSR-Befehle) abweichen.

Der Benutzer des Rahmens darf keine START-, ALARM-, XBASIS-, UNTPR- und VORBES-Befehle schreiben.

Da das Montageobjekt S&CC angeschlossen ist, stehen dem Rahmenbenutzer auch die übrigen Möglichkeiten dieser Kontrollprozedur zur Verfügung (Abschlussbehandlung, Zugriff zu einigen Konstanten wie etwa 1 und pi).

4. Makros für die Eingabe

Diese Makros dienen zur Eingabe von Daten aus dem DATEN-Fremdstring des STARTE- (oder RECHNE-) Kommandos.

4.1. Das Makro R&LIES

```
⟨Makro R&LIES⟩  ::=  R&LIES [(⟨16-Bit-Sprungadresse nach Stringende⟩)]
```

Voraussetzung: Bei Verwendung dieses Makros wird normalerweise im STARTE- (oder RECHNE-) Kommando ein DATEN-Fremdstring angegeben sein. Das Zeichen / der DATEN-Spezifikation muß entweder am Ende einer Zeile (=ausgenutzter Teil einer Lochkarte) stehen oder dem Zeichen / dürfen in derselben Zeile nur Blanks folgen. Nur die darauf folgenden Zeilen (möglicherweise 0 Zeilen) bis vor das nächste Fluchtsymbol gelten dann als Fremdstring.

Wirkung: Dieses Makro liest aus dem DATEN-Fremdstring des STARTE (oder RECHNE-) Kommandos ungeachtet einer evtl. Kartennumerierung die nächste Zeile in Form von Zentralcode-Oktaden (pro Halbwort eine Oktade und zwar rechtsbündig und mit NUL-Zeichen aufgefüllt) in den Eingabepuffer ab Halbwort R&EPUFFER und speichert im Register A die Anzahl der eingelesenen Oktaden. Der Eingabepuffer-Zeiger wird auf das erste Halbwort des Eingabepuffers gerichtet. Die in den Eingabepuffer gelieferte Information kann dann beliebig weiter bearbeitet werden, z.B. durch Wortgruppentransport oder mit den Makros von Kapitel G.4.3.

Falls im STARTE- (oder RECHNE-) Kommando kein DATEN-Fremdstring angegeben ist oder falls schon vor dem betreffenden Aufruf des Makros der Fremdstring vollständig gelesen war, wird innerhalb derselben Großseite auf den im Makroaufruf angegebenen Platz gesprungen; bei Fehlen einer Sprungadresse werden der Fehlertext

R&LIES FINDET WEDER DATEN NOCH SPRUNGADRESSE

und der Fehlerort ins Ablaufprotokoll gedruckt und der Operatorlauf abgebrochen (genaueres in Kapitel G.2).

In einigen Fällen wird der Fehler

VERMUTUNG: FUER DATEN FALSCHE CODE-EINSTELLUNG
(Z.B. BINAER) ODER ZEILE MIT UEBER 80 ZEICHEN

gemeldet und der Operatorlauf abgebrochen.

Erweiterung für Sonderfälle:

Daten können statt als DATEN-Fremdstring auch durch die Spezifikation

DATEI = 98-[⟨Datenbasisname⟩.]⟨Dateiname⟩

des STARTE-Kommandos verfügbar gemacht werden. Die Sätze dieser Datei dürfen höchstens 80 Zeichen lang sein.

Beispiel

```
LESEN=      R&LIES(WEITER),
   •
   •
   •
            S     LESEN,
WEITER              --SPRUNG NACH STRINGENDE--
```


4.2. Der Eingabepuffer-Zeiger

Der Standardrahmen führt einen Zeiger (R&EPUFFERPOS) ein, der (in Form absoluter Adressen) angibt, von welchem Halbwort des Eingabepuffers ab die nächsten Oktaden zu verarbeiten sind (pro Halbwort ist eine Oktade gespeichert). Der Zeiger zeigt nach Ausführung des Makros R&LIES auf das erste Halbwort des Eingabepuffers. Der Zeiger kann durch das Makro R&EPOS auf eine anzugebende Position im Eingabepuffer gesetzt oder durch das Makro R&ESKIP um eine anzugebende Anzahl von Halbworten weitergerückt werden. Außerdem wird er durch die Makros von Kapitel G.4.3 weitergerückt, und zwar um die Anzahl von Halbworten, die das betreffende Makro verarbeitete.

4.2.1. Das Makro R&EPOS

⟨Makro R&EPØS	::=	R&EPØS (⟨n⟩)
⟨n⟩	::=	(natürliche Zahl, wobei 1 ≤ n ≤ 80)

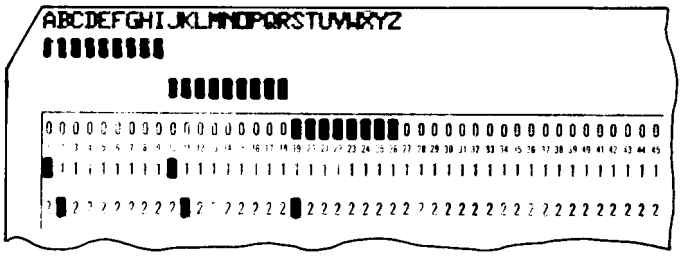
Der Eingabepufferzeiger wird auf das n-te Halbwort des Eingabepuffers gerichtet (das die Adresse R&EPUFFER+n-1 hat).

4.2.2. Das Makro R&ESKIP

⟨Makro R&ESKIP⟩	::=	R&ESKIP [(⟨n⟩)]
⟨n⟩	::=	(natürliche Zahl, wobei 1 ≤ n ≤ 79)

Der Eingabepuffer-Zeiger wird um n Halbworte weitergerückt. Ist für n keine Angabe gemacht, so wird für n=1 angenommen.

Beispiel



Von der LK sollen die Buchstaben CD und KL eingelesen werden.

```
R&LIES(R&ENDE),
R&EPOS(3)          --DER EINGABEPUFFERZEIGER WIRD AUF
                     SPALTE 3 GESETZT--
R&EOKT(2)          --EINLESEN DER SPALTEN 3 UND 4--
C      NAME,
R&ESKIP(6)         --DER ZEIGER WIRD AUF SPALTE 10 GESETZT--
R&EOKT(2)          --EINLESEN DER SPALTEN 11 UND 12--
C      NOME,
```

4.3. Makros zum Manipulieren von Eingabeinformation

Diese Makros dienen zum Herausschneiden, Umschlüsseln und Konvertieren von Information aus dem Eingabepuffer, die z.B. durch R&LIES dorthin kam.
Es ist Sache des Benutzers, darauf zu achten, daß nur die durch R&LIES (oder sonstwie) definierte Information bearbeitet wird.

Undurchführbare Anweisungen werden wie folgt behandelt: Bei unzulässigem Argument wird eine Fehlermeldung ausgegeben und der Operatorlauf beendet (genaueres in Kapitel G.2). Bei falschen Eingabedaten wird ohne Fehlermeldung auf die im Makroaufruf evtl. angegebene 16-Bit-Fehleradresse gesprungen, anderenfalls eine der folgenden Fehlermeldungen gegeben

UNZULAESSIGES ZEICHEN
ZU GROSSE ZAHL
FEHLERHAFTE EINGABEDATEN

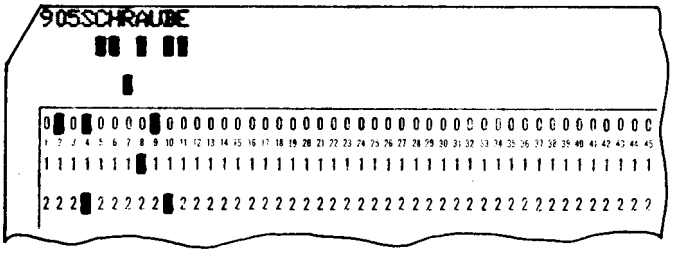
und der Operatorlauf abgebrochen.

4.3.1. Das Makro R&EOKT

⟨Makro R&EØKT⟩	::=	R&EØKT [(⟨n⟩)]
⟨n⟩	::=	(natürliche Zahl, wobei 1 ≤ n ≤ 6)

Eine Zeichenfolge von n Oktaden wird gepackt, linksbündig in das Register A gebracht, mit NUL-Oktaden aufgefüllt und mit TK3 versehen.
Wird für n keine Angabe gemacht, so wird für n=1 angenommen.
Nach der Bearbeitung wird der Eingabepuffer-Zeiger um n Halbworte weitergerückt. Der Eingabepuffer bleibt unverändert.

Beispiel



Die Lochkarte soll eingelesen werden und das Wort Schraube in ein Ganzwort gelegt werden.

```
R&LIES,
R&EOKT(3),
C      KA,
R&EOKT(6)          --EINLESEN DES TEXTES SCHRAU--
C      BEZ,
R&EOKT(2),         --EINLESEN DES TEXTES BE--
C      BEZ+2,
```

4.3.2. Das Makro R&ETET

⟨Makro R&ETET⟩	::=	R&ETET [((n)[,⟨F-Adr.⟩])]
⟨n⟩	::=	(natürliche Zahl, wobei 1 ≤ n ≤ 12)
⟨F-Adr.⟩	::=	, 16-Bit-Sprungadresse

Bei Angabe von R&ETET (n) wird die Zeichenfolge des Eingabepuffers in eine Tetradenfolge umgeschlüsselt, rechtsbündig in das Register A gebracht, mit 0-Tetraden aufgefüllt und mit TK2 versehen. Dabei werden führende SP- (Blank) Oktaden in 0-Tetraden umgeschlüsselt.

Die Zahl n gibt die Anzahl der Halbworte des Eingabepuffers an, die umgeschlüsselt werden sollen. Gleichzeitig wird der Eingabepuffer-Zeiger um n Stellen weitergerückt. Wird für n keine Angabe gemacht, so wird n=1 angenommen.

Das Makro kann mit einer Sprungadresse versehen werden, die bei Auftreten eines nicht verarbeitbaren Zeichens angesprungen wird.

Nach Ausführung der Leistung des Makros bleibt die Zeichenfolge im Eingabepuffer unverändert.

Beispiel

```
R&ETET(8,FADR),
```

Es werden 8 Halbworte des Eingabepuffers umgeschlüsselt. Beim Auftreten eines nicht verarbeitbaren Zeichens wird die Adresse FADR angesprungen.

4.3.3. Das Makro R&EGANZ

```
⟨Makro R&EGANZ⟩  ::=  R&EGANZ (⟨n⟩ [,⟨F-Adr.⟩],)
⟨n⟩              ::=  (natürliche Zahl, wobei 1 ≤ n ≤ 80)
⟨F-Adr.⟩              siehe Kapitel G.3.2
```

Bei Angabe von R&EGANZ (n) wird die Zeichenfolge des Eingabepuffers in eine binäre Festpunktzahl konvertiert, rechtsbündig im Register A abgelegt und mit TK1 versehen.
Die Zahl n gibt an, wieviele Halbworte des Eingabepuffers verarbeitet werden sollen. Gleichzeitig wird der Eingabepuffer-Zeiger um n erhöht.
Die Oktadenfolge muß die Form

$$\text{ƀ}^{0-\infty}\,[-\text{ƀ}^{0-\infty}]\cdot\langle\text{Ziffer}\rangle^{1-13}$$

haben.

Der Inhalt des Eingabepuffers bleibt nach Ausführung des Makros unverändert.
Das Makro kann mit einer Fehleradresse versehen werden.
Zahlen, die durch R&EGANZ eingelesen werden, können anstelle der Ziffer 0 auch eine Leerstelle haben. Die Zahl muß dann mindestens eine Ziffer enthalten.

4.3.4. Das Makro R&EBRUCH

```
⟨Makro R&EBRUCH⟩  ::=  R&EBRUCH (⟨n⟩ [,⟨F-Adr.⟩])
⟨n⟩               ::=  (natürliche Zahl, wobei 1 ≤ n ≤ 80)
⟨F-Adr.⟩               siehe Kapitel G.3.2
```

Bei Angabe von R&EBRUCH (n) wird die Zeichenfolge des Eingabepuffers in eine binäre Festpunktzahl umgeschlüsselt und zwar im Register A linksbündig mit TK1.
Die Zahl n gibt die Anzahl der Halbworte des Eingabepuffers an, die umgeschlüsselt werden sollen. Gleichzeitig wird der Eingabepuffer-Zeiger um n erhöht. Der Eingabepuffer bleibt unverändert.
Die Oktadenfolge muß die Form

$$\text{ƀ}^{0-\infty}\,[-\text{ƀ}^{0-\infty}]\cdot\langle\text{Ziffer}\rangle^{1-13}$$

haben.

Der absolute maximale Fehler des Resultats hat die Größenordnung 2^{-43}.
Das Makro kann mit einer Fehleradresse versehen werden.

4.3.5. Das Makro R&EGLEIT

⟨Makro R&EGLEIT⟩	::=	R&EGLEIT (⟨n⟩[,⟨F-Adr.⟩])
⟨n⟩	::=	(natürliche Zahl, wobei 4 ≤ n ≤ 80)
⟨F-Adr.⟩		siehe G.4.3.2

Bei Angabe von R&EGLEIT wird die Zeichenfolge des Eingabepuffers in eine Gleitpunktzahl einfacher Genauigkeit umgeschlüsselt, und zwar im Register A mit TK0.
Die Zahl n gibt die Anzahl der Halbworte des Eingabepuffers an, die umgeschlüsselt werden sollen. Der Eingabepuffer-Zeiger muß zuvor auf die erste zu interpretierende Oktade gesetzt werden. Der Eingabepuffer bleibt unverändert.
Die Oktadenfolge muß die Form

$$ƀ^{0-\infty}\,[-ƀ^{0-\infty}]\,.\,\langle\text{Ziffer}\rangle^{1-\infty}\,ƀ^{0-\infty}\,Eƀ^{0-\infty}\,[-ƀ^{0-\infty}]\,\langle\text{Ziffer}\rangle^{1-3}$$

haben.
Die Gleitpunktzahl hat einen maximalen absoluten Fehler von der Größenordnung

$$2^{-38}\cdot 16^{\text{Sedezimalexponent}}$$

Das Makro kann mit einer Fehleradresse versehen werden.

4.3.6. Das Makro R&WIED

⟨Makro R&WIED⟩	::=	R&WIED (⟨n⟩, ⟨Makro-Konstante⟩)
⟨n⟩	::=	(natürliche Zahl, wobei ≤ n ≤ 52)
⟨Makro-Konstante⟩		siehe Kapitel D.3

Bei Angabe von R&WIED wird Element n-mal durch ein Komma zu einer Informationseinheit ergänzt.

Beispiel

Der Aufruf

```
R&WIED(2,''VORBES''/V),
```

liefert

```
''VORBES''/V,''VORBES''/V,
```

G

Sonderfälle

Die globale Makrovariable R&NR enthält die Angabe, zum wievielten Mal durch diesen Aufruf momentan obige Informationseinheit erzeugt wird.

Beispiel

Der Aufruf

```
DEF             IE,
NR+(R&NR)=      A+1/A,
DEND,
R&WIED          (3,IE),
```

liefert

```
NR1=   A+1/A,  NR2=   A+1/A,  NR3=   A+1/A,
```

- Will man aus dem Rahmen nur das Makro R&WIED verwenden, so gibt man im UEBERSETZE-Kommando die Spezifikation SPRACHE = TAS an und schreibt vor dem ersten Aufruf von R&WIED:

  HDEF&OEFDB (R&RAHMEN, R&WIED),

Anmerkung: Die Leistung von R&WIED wird durch den Pseudobefehl REPL abgedeckt (ohne die Einschränkung n $\leq$52).

5. Makros für die Ausgabe

5.1. Der Ausgabepuffer-Zeiger

Der Standardrahmen führt einen Zeiger (R&APUFFERPOS) ein, der (in Form absoluter Adressen) angibt, von welchem Halbwort des Ausgabepuffers ab die nächsten Oktaden zu speichern sind (pro Halbwort wird eine Oktade gespeichert). Der Zeiger zeigt bei Start des Programms wie auch nach Ausführung des Makros R&DRUCKE auf das erste Halbwort des Ausgabepuffers. Der Zeiger kann durch das Makro R&APOS auf eine anzugebende Position im Ausgabepuffer gesetzt werden oder durch das Makro R&ASKIP um eine anzugebende Anzahl von Halbworten weitergerückt werden. Außerdem wird er durch die Makros von Kapitel G.5.2 weitergerückt, und zwar um die Anzahl von Halbworten, die zum Speichern der Oktadenfolge erforderlich ist, die das betreffende Makro erzeugt. Normalerweise ist pro Halbwort des Ausgabepuffers ein Druckzeichen (oder Leerzeichen) gespeichert; ist diese Voraussetzung in Sonderfällen nicht erfüllt, so besteht im allgemeinen kein einfacher Zusammenhang mehr zwischen dem Zeiger des Ausgabepuffers und der Position auf dem Ablaufprotokoll.

5.1.1. Das Makro R&APOS

⟨Makro R&APØS⟩	::=	R&APØS (⟨n⟩)
⟨n⟩	::=	(natürliche Zahl, wobei $1 \leq n \leq 116$)

Der Ausgabepuffer-Zeiger wird auf das n-te Halbwort des Ausgabepuffers gerichtet (das die Adresse R&APUFFER+n-1 hat).

5.1.2. Das Makro R&ASKIP

⟨Makro R&ASKIP⟩	::=	R&ASKIP [(⟨n⟩)]
⟨n⟩	::=	(natürliche Zahl, wobei 1 ≤ n ≤ 116)

Der Ausgabepuffer-Zeiger wird um n Halbworte weitergerückt. Ist für n keine Angabe gemacht, so wird für n=1 angenommen.

5.2. Makros zum Zusammensetzen von Ausgabeinformation

Diese Makros dienen, evtl. nach Umschlüsselung und Konvertierung, zum Einfügen von Information in den Ausgabepuffer, dessen Inhalt später mit Hilfe von R&DRUCKE ausgedruckt werden kann.

5.2.1. Das Makro R&AOKT

⟨Makro R&AØKT⟩	::=	R&AØKT [(⟨n⟩)]
⟨n⟩	::=	(natürliche Zahl, wobei 1 ≤ n ≤ 6)

Die Zahl n gibt an, wieviele Oktaden aus dem A-Register für die Ausgabe bereitgestellt werden sollen. Die n linksersten Oktaden des A-Registers werden ohne Prüfung auf Druckfähigkeit in den Ausgabepuffer übertragen. Hierbei ist zu beachten, daß auf diese Weise z.B. auch NUL-Oktaden in den Ausgabepuffer an Stellen kommen, an denen normalerweise Druckzeichen (oder Leerstellen) stehen, sodaß sich die Wirkung von R&APOS und R&ASKIP ändert. Die Typenkennung im A-Register bleibt unbeachtet.
Wird für n kein Wert angegeben, wird n=6 angenommen.

5.2.2. Das Makro R&ATET

```
⟨Makro R&ATET⟩  ::=  R&ATET [(⟨n⟩)]
⟨n⟩             ::=  (natürliche Zahl, wobei 1 ≤ n ≤ 12)
```

Bei Angabe von R&ATET werden die n rechtsletzten Tetraden aus dem A-Register genommen, in Oktaden umgeschlüsselt und im Ausgabepuffer halbwortweise abgelegt. Die Typenkennung des A-Registers bleibt unbeachtet.
Wird für n keine Angabe gemacht, so wird n=12 angenommen.

5.2.3. Das Makro R&AGANZ

```
⟨Makro R&AGANZ⟩  ::=  R&AGANZ [(⟨n⟩)]
⟨n⟩              ::=  (natürliche Zahl, wobei 1 ≤ n ≤ 14)
```

Bei Angabe von R&AGANZ wird der Inhalt des A-Registers als rechtsbündige Festpunktzahl interpretiert und in eine Oktadenfolge der Form

$$\left\{ \begin{matrix} \text{b} \\ - \end{matrix} \right\} \langle \text{Ziffer} \rangle^{n-1}$$

konvertiert. Die Oktaden werden im Ausgabepuffer halbwortweise abgelegt. Die Zahl n gibt an, wieviele Halbworte des Ausgabepuffers belegt, bzw. wieviele Druckstellen in Anspruch genommen werden sollen. Führende Nullen werden durch Leerzeichen ersetzt, ausgenommen in der rechtsletzten Stelle der Zahl. Die Stelle vor der ersten ausgedruckten Ziffer bleibt dem Vorzeichen vorbehalten. Bei positiven Zahlen als Leerzeichen (Blank, b), bei negativen Zahlen (einschließlich der negativen Null) erscheint das Minuszeichen. Es werden also höchstens n-1 Ziffern gedruckt.
Wird für n keine Angabe gemacht, so wird n=14 angenommen.
Kann der Betrag der auszugebenden Zahl nicht mit n-1 Dezimalstellen dargestellt werden, so wird die Fehlermeldung "ZU GROSSE ZAHL" gegeben und der Operatorlauf abgeschlossen.

5.2.4. Das Makro R&ABRUCH

⟨Makro R&ABRUCH⟩ ::= R&ABRUCH [(15)]

Bei Angabe dieses Makros wird der Inhalt des A-Registers als linksbündige Festpunktzahl interpretiert und (mit Hilfe des Befehls KFLD) in eine 15-stellige Oktadenfolge der Form

$$\left\{\begin{matrix}\text{ƀ}\\-\end{matrix}\right\}\langle\text{Ziffer}\rangle^{13}$$

konvertiert. Die Oktaden werden halbwortweise im Ausgabepuffer abgelegt. Von den 15 Oktaden ist die erste dem Vorzeichen vorbehalten. Bei positiven Zahlen als Leerzeichen (Blank, ƀ), bei negativen Zahlen (einschließlich der negativen Null) erscheint das Minuszeichen. Eine weitere Oktade enthält den Dezimalpunkt. Für Ziffern bleiben demzufolge noch 13 Oktaden übrig.
Die Typenkennung des A-Registers bleibt unbeachtet. Mögliche Fehlermeldung während der Übersetzung:

R&ABRUCH HAT FALSCHES ARGUMENT.

Der maximale absolute Fehler der konvertierten Zahl ist von der Größenordnung 10^{-13}.

5.2.5. Das Makro R&AGLEIT

⟨Makro R&AGLEIT⟩ ::= R&AGLEIT [(19)]

Bei Angabe des Makro R&AGLEIT wird der Inhalt des A-Registers als Gleitpunktzahl interpretiert und in eine 19-stellige Oktadenfolge der Form

$$\left\{\begin{matrix}\text{ƀ}\\-\end{matrix}\right\}\langle\text{Ziffer}\rangle^{12}\ \text{E}\ \left\{\begin{matrix}\text{ƀ}\\-\end{matrix}\right\}\langle\text{Ziffer}\rangle^{3}$$

konvertiert. Die Typenkennung des A-Registers bleibt unbeachtet. Mögliche Fehlermeldung während der Übersetzung:

R&AGLEIT HAT FALSCHES ARGUMENT.

Der maximale, absolute Fehler der konvertierten Zahl ist von der Größenordnung

$$10^{-2}\cdot 10^{\text{Dezimalexponent}}$$

Die so erzeugte Oktadenfolge wird in dasjenige Halbwort und die nachfolgenden gespeichert, auf das der Ausgabepuffer-Zeiger zeigt; gespeichert wird in gespreizter Form (d.h. NUL NUL Oktade pro Halbwort), und zwar nur, soweit gemäß dem Ausgabepuffer-Zeiger noch Platz im Ausgabepuffer ist. Überschüssige Information geht verloren. Nach dem Abspeichern wird der Ausgabepuffer-Zeiger um n Halbworte (meist gleich der Anzahl der erzeugten druckbaren Oktaden) weitergerückt.

5.3. Das Makro R&DRUCKE

```
⟨Makro R&DRUCKE⟩  ::=  R&DRUCKE
```

Dieses Makro druckt die im Ausgabepuffer vorliegende Folge von Zentralcode-Oktaden in das Ablaufprotokoll (bei Gesprächen auch in das Fernschreiberprotokoll). Der Puffer kann z.B. durch Wortgruppentransport oder durch die Makros von Kapitel G.5.2 gefüllt worden sein. Die Zeilenlänge beträgt 116 Zeichen (unabhängig vom Ausgabepuffer-Zeiger); darüber hinausgehende Zeichen gehen verloren. Der Papiervorschub vor dem Drucken richtet sich nach der linksersten Oktade in R&VORSCHUB; diese Oktade kann man mit Hilfe des Makros R&SEITE oder R&VORSCHUB wählen, wenn man vom Standard abweichen will; Standard ist der Vorschub auf die nächste Zeile und nach 63 Zeilen der Vorschub auf die nächste Seite. Nach dem Drucken werden alle Halbworte des Ausgabepuffers auf NUL NUL SP gesetzt, als Papiervorschub für einen späteren Aufruf von R&DRUCKE "nächste Zeile" vorgesehen und der Ausgabepuffer-Zeiger auf das erste Halbwort des Ausgabepuffers gerichtet.

Erweiterung für Sonderfälle:

Will man durch R&DRUCKE nicht ins Ablaufprotokoll, sondern in eine Datei ausgeben (etwa um die Kopfzeile des Ablaufprotokolls zu vermeiden oder um das Protokoll vervielfältigen zu können), so gibt man im STARTE-Kommando die Spezifikation

DATEI = 99 - [⟨Datenbasisname⟩ .] ⟨Dateiname⟩

an. Die Datei muß zuvor deklariert sein mit SATZBAU=A und einer Längenangabe, die für 354 Satzelemente ausreicht. Evtl. Fehlermeldungen und Binärdumps werden weiterhin in das Ablaufprotokoll gedruckt.

5.3.1. Das Makro R&DRUCKETEXT

⟨Makro R&DRUCKETEXT⟩	::=	R&DRUCKETEXT (⟨Oktadenfolge⟩)
⟨Oktadenfolge⟩		siehe Kapitel B.4.6

Durch die Angabe von R&DRUCKETEXT wird der Ausgabepuffer gelöscht und die Oktadenfolge ohne die einleitenden und schließenden Apostrophe in den Ausgabepuffer gespeichert. Anschließend wird das Makro R&DRUCKE ausgeführt.

Beispiel

```
R&DRUCKETEXT(''PROGRAMM-PUNKT 2 ERREICHT''),
```

5.4. Das Makro R&DRUCK

⟨Makro R&DRUCK⟩	::=	R&DRUCK

Die Angabe von R&DRUCK wirkt wie der Makro-Aufruf R&DRUCKE, jedoch werden die eventuell am Ende des Puffers stehende Leerzeichen nicht mit ausgegeben. Dadurch verringert sich die Ausgabezeit bei Fernschreibern. Im Gegensatz zu R&DRUCKE wird der Ausgabepuffer (normalerweise 116 Zeichen) nicht generell in zwei Zeilen gedruckt, sondern bei einer Zeichenzahl von ≤ 69 nur in eine Zeile.

5.4.1. Das Makro R&DRUCKTEXT

⟨Makro R&DRUCKTEXT⟩	::=	R&DRUCKTEXT (⟨Oktadenfolge⟩)
⟨Oktadenfolge⟩		siehe Kapitel B.4.6

Durch die Angabe von R&DRUCKTEXT wird der Ausgabepuffer gelöscht und die Oktadenfolge ohne die einleitenden und schließenden Apostrophe in den Ausgabepuffer gespeichert. Anschließend wird das Makro R&DRUCK ausgeführt.

5.5. Das Makro R&SEITE

```
⟨Makro R&SEITE⟩  ::=  R&SEITE
```

Dieses Makro sorgt dafür, das am Anfang des nächsten Aufrufs von R&DRUCKE das Ablaufprotokoll auf den nächsten Seitenanfang vorgeschoben wird.

5.6. Das Makro R&VORSCHUB

```
⟨Makro R&VØRSCHUB⟩  ::=  R&VØRSCHUB [(⟨n⟩)]
⟨n⟩                 ::=  (natürliche Zahl, wobei 1 ≤ n ≤ 6)
```

Dieses Makro sorgt dafür, das am Anfang des nächsten Aufrufs von R&DRUCKE das Ablaufprotokoll um n Zeilen vorgeschoben wird.
Wird für n keine Angabe gemacht, so wird n=1 angenommen.

Zusatz: Die Wirkung eines Vorschubs mit n >1, welcher über den standardmäßigen 63-Zeilen-Raum einer Seite hinausreicht, ist nicht festgelegt.

5.7. Das Makro R&DUMP

```
⟨Makro R&DUMP⟩      ::=  R&DUMP [⟨TAS-DUMP⟩]
⟨TAS-DUMP⟩          ::=  { ("ALLES  } (⟨Variablenname⟩[ ,⟨Variablenname⟩]^∞)")
                         { ("NICHTS }
⟨Variablenname⟩     ::=  ⟨Name⟩
⟨Name⟩                   siehe Kapitel B.4.1
```

Bei Angabe dieses Makro wird ein TAS-Variablen-Dump geliefert ohne Abbruch der Programmausführung. Die Angaben zu TAS-Dump sind identisch mit denen, die im Kommando-Handbuch, Kapitel 3, STARTE-Kommando, Spezifikation DUMP stehen. Allerdings ohne Leerstellen und die einleitenden Zeichen T- .
Wird bei R&DUMP kein Argument angegeben, so wird die im STARTE-Kommando stehende Spezifikation als maßgebend angenommen und R&DUMP wirkt wie ein NULL-Befehl.

Beispiel

```
R&DUMP(''NICHTS(R&INDZELLEN)''),
```

5.8. Das Makro R&BINAERDUMP

⟨Makro R&BINAERDUMP⟩	::=	R&BINAERDUMP (⟨Anfangsadresse⟩,⟨Endadresse⟩)
⟨Anfangsadresse⟩, ⟨Endadresse⟩	::=	16-Bit oder 22-Bit-Adresse, die als Ganzwortadressen gedeutet werden.

Es wird die im angegebenen Adreßbereich vorliegende Information in Form von Tetraden ins Ablaufprotokoll gedruckt; auch die zugehörigen Adressen werden gedruckt.

Zusatzregel: Durch den Makroaufruf

```
R&BINAERDUMP(R&EPUFFER,R&EPUFFER+199),
```

erhält man eine Ausschrift derjenigen Variablen, die im Standardrahmen durch folgende Deklarationen (in dieser Reihenfolge) eingeführt werden:

```
R&EPUFFER.        ASP 80/G,
R&EPUFFERPOS.     ASP 1,
R&APUFFERPOS.     ASP 1,
R&VORSCHUB.       ASP 2/G,
R&APUFFER.        ASP 116/G,
```

6. Makros zum Beenden der Programmausführung

6.1. Das Makro R&FEHLER

⟨Makro R&FEHLER⟩	::=	R&FEHLER (⟨Oktadenkonstante⟩)
⟨Oktadenkonstante⟩		siehe Kapitel B.10.5

Die Oktadenkonstante darf maximal 100 Zeichen lang sein (damit paßt der Text in eine Zeile, vergl. Kapitel G.5.3).

Bei Angabe von TRACE =-STD- im UEBERSETZE-Kommando werden die Befehle

TEST*EIN, TEST* AUS,

ausgeführt.

In die nächste Zeile des Ablaufprotokolls werden gedruckt: die durch die Oktadenkonstante dargestellte Zeichenfolge und eine Angabe über die Adresse eines der vom Makroaufruf erzeugten Befehle.

Gegebenenfalls fólgt ein Variablen-Dump.

Der Operatorlauf wird beendet und das nächste Kommando bearbeitet.

Beispiel

Der Aufruf

```
R&FEHLER(''UNERWARTETER FALL''),
```

liefert z.B. folgenden Ausdruck

```
UNERWARTETER FALL FEHLERORT: 00142C
```

6.2. Das Makro R&ENDE

⟨Makro R&ENDE⟩ ::= R&ENDE

Bei Angabe von TRACE =-STD- im UEBERSETZE-Kommando wird der Befehl TEST*AUS, ausgeführt.

Der Operatorlauf wird beendet und das nächste Kommando bearbeitet.

6.3. Die Adresse R&ENDE

Unter dem globalen Namen R&ENDE ist die 16-Bit-Adresse derselben Befehlsfolge verfügbar, die auch durch das Makro R&ENDE (siehe Kapitel G.6.2) aufgerufen wird.

Beispiel

Die Informationseinheit SIO R&ENDE, beendet den Operatorlauf, falls ⟨A⟩ = 0 ist.

6.4. Die Adresse R&SSRFEHLER

Als Fehleradresse im Versorgungsblock eines SSR-Befehls kann der globale Name R&SSRFEHLER angegeben werden; dann geschieht im Fehlerfall folgendes:

Aufgrund der Registerinhalte wird eine Fehlernachricht gegeben, die etwa so aussieht:

```
FEHLER NR. 56 BEI SSR 253 10
AUFRUF VON ADR. 003650
<Q> = 3 000000000000
<H> = 2 00000000001F
```

Gegebenenfalls folgt ein Variablendump.

Der Operatorlauf wird beendet und das nächste Kommando bearbeitet.

7. Beispiel

Bei Verwendung der Spezifikation SPRACHE = TASR im UEBERSETZE-Kommando bilden die folgenden Zeilen bereits ein vollständiges TAS-Programm:

```
ANFANG=     NULL  0,
          --LIES DIE NAECHSTE LOCHKARTE--
            R&LIES(R&ENDE),
          --SPEICHERE AUS DEN 2 ERSTEN SPALTEN
            GANZE ZAHL NACH V1--
            R&EGANZ(2),
V1=         ASP   2/G,
            C     V1,
          --SPEICHERE AUS DER NAECHSTEN SPALTE
            GANZE ZAHL NACH V2--
            R&EGANZ(1),
V2=         ASP   2/G,
            C     V2,
          --SETZE V3=V1+V2--
            A     V1,
V3=         ASP   2/G,
            C     V3,
          --DRUCKE V1 PLUS V2=V3--
            B     V1,
            R&AGANZ,
            B     (''PLUS''),
            R&AOKT(6),
            B     V2,
            R&AGANZ,
            B     (''=''),
            R&AOKT(3),
            B     V3,
            R&AGANZ,
            R&DRUCKE,
          --SPRINGE NACH ANFANG--
            S     ANFANG,
            ENDE,
```

TESTHILFEN

1.	Der Überwacher (TEST*BEFEHLE)	1
2.	Die Testbefehle	2
2.1.	TEST*EIN-Befehl	3
2.2.	TEST*AUS-Befehl	3
2.3.	TEST*SR-Befehl	3
2.4.	TEST*SD-Befehl	4
2.5.	TEST*FR-Befehl	4
2.6.	TEST*FD-Befehl	4
2.7.	TEST*CR-Befehl	5
2.8.	TEST*XR-Befehl	5
2.9.	TEST*BR-Befehl	5
2.10.	TEST*SPRUNG-Befehl	6
2.11.	TEST*4XR-Befehl	6
2.12.	TEST*FORT-Befehl (Überwachung des SSR-Befehls)	6
2.13.	TEST*SSATZ-Befehl	7
2.14.	TEST*DATEI-Befehl	7
3.	Allgemeines zu den Testbefehlen	8
4.	Kontrollereignisse	8
4.1.	KE-Befehl	8

H

TESTHILFEN

1. Der Überwacher (TEST*BEFEHLE)

Der Überwacher stellt eine Reihe sogenannter Testbefehle für das Austesten von übersetzten TAS-Quellenprogrammen zu Verfügung, die im Normalmodus laufen sollen. Die Testbefehle sind Anweisungen an den Überwacher und werden während des Ablaufs des Programms ausgeführt. Sie veranlassen den Überwacher dazu, die Ausführungen von TR 440-Befehlen zu überwachen und zusätzlich abhängig von bestimmten Bedingungen Registerstände und Speicherinhalte auszudrucken.

H

Die Testbefehle werden im TAS-Quellenprogramm in der abgekürzten Form (ohne Befehlscode z. B. TEST*EIN) angegeben. Sie werden vom TAS-Assembler in SUE-Befehle übersetzt. Zu den meisten Testbefehlen gehört noch (anschließend) die Adresse eines Versorgungsblocks, welcher an beliebiger Stelle stehen kann. (Soll der Versorgungsblock in Großseite 0 stehen, so benutzt man am besten die Literalschreibweise: NULL (⟨Versorgungsblock⟩).)

Die Benennung eines Test-Befehls wird dem SUE-Befehl gegeben oder, bei TRACE =-, an die nächste Informationseinheit.

Die in einem Quellenprogramm enthaltenen Testbefehle (einschließlich der meist folgenden Adressenkonstanten bzw. Nullbefehle) werden vom Assembler genau dann überlesen, wenn die Spezifikation TRACE im UEBERSETZE-Kommando nicht definiert ist (siehe Kapitel E. 8).

Der Überwacher wird bei der Montage an das zu überwachende Quellenprogramm angeschlossen.

2. Die Testbefehle

Es gibt folgende Testbefehle

TEST*EIN	Einschalten des Überwachers
TEST*AUS	Ausschalten des Überwachers
TEST*SR	Drucke sofort die Registerstände der als nächstes ausgeführten n Befehle
TEST*SD	Gib sofort einen Dump der angegebenen Speicher- und Indexbereiche
TEST*FR	Falls der Befehlszähler in einem der angegebenen Bereiche ist, drucke die Registerstände
TEST*FD	Gib bei Erreichen eines Befehlszählerstandes einen Dump der angegebenen Speicher- und Indexbereiche
TEST*CR	Falls ein Wort aus einem der angegebenen Speicherbereiche angesprochen wird, drucke die Registerstände
TEST*XR	Falls eine Indexzelle aus einem der angegebenen Index- oder Speicherbereiche angesprochen wird, drucke die Registerstände
TEST*BR	Falls der Befehlscode einer der angegebenen ist, drucke die Registerstände
TEST*SPRUNG	Falls Sprungbefehl, drucke die Registerstände
TEST*4XR	Simuliere die 4 Indexregister
TEST*FORT	Fortsetzen nach einem SSR-Befehl
TEST*SSATZ	Angaben der Startsatzadresse (für Seitenschranke)
TEST*DATEI	Angabe einer Druckdatei zur Aufnahme des Überwacherprotokolls

2.1. TEST*EIN-Befehl

```
⟨TEST*EIN-Befehl⟩  ::=  TEST*EIN
```

Der TEST*EIN-Befehl bewirkt das dynamische Einschalten des Überwachers und gleichzeitigen Druck der Registerstände. Der eingeschaltete Überwacher überwacht alle auszuführenden TR 440-Befehle und nimmt weitere Testbefehle entgegen. Die Überwacherschranke im Startsatz wird ausgewertet.

2.2. TEST*AUS-Befehl

```
⟨TEST*AUS-Befehl⟩  ::=  TEST*AUS
```

Der TEST*AUS-Befehl bewirkt sofortiges Ausschalten des Überwachers und gleichzeitigen Druck der Registerstände. Vom Überwacher angesammelte Druckinformation wird ausgegeben.

2.3. TEST*SR-Befehl

```
⟨TEST*SR-Befehl⟩  ::=  TEST*SR, ⟨n⟩/AB [M] [N]
⟨n⟩               ::=  ⟨natürliche Zahl⟩
```

Der TEST*SR-Befehl bewirkt den Druck der Registerstände bei den nächsten n Befehlen. Falls die Versorgungsadreßkonstante N-markiert ist, wird anschließend der Überwacher ausgeschaltet.

2.4. TEST*SD-Befehl

```
⟨TEST*SD-Befehl⟩  ::=  TEST*SD, ⟨V-Block-Adr.⟩/AB
```

Der TEST*SD-Befehl weist den Überwacher an, den Inhalt von Speicher- und/oder Indexbereichen auszudrucken. Der Versorgungsblock besteht aus Adreßkonstanten-Paaren, die den jeweiligen Speicherbereich (unmarkierte Adreßkonstanten) bzw. Indexbereich (N-markierte Adreßkonstanten) angeben.

2.5. TEST*FR-Befehl

```
⟨TEST*FR-Befehl⟩  ::=  TEST*FR, ⟨V-Block-Adr.⟩/AB [M]
```

Der TEST*FR-Befehl weist den Überwacher an, die Registerstände auszudrucken, wenn der Befehlszählerstand in einem der angegebenen Bereiche liegt. Der Versorgungsblock besteht aus Adreßkonstanten-Paaren, die den jeweiligen Befehlszählerbereich angeben. Soll ein einzelner Befehlszählerstand überwacht werden, so wird anstelle eines Adreßkonstanten-Paares eine M-markierte Adreßkonstante angegeben.

2.6. TEST*FD-Befehl

```
⟨TEST*FD-Befehl⟩  ::=  TEST*FD, ⟨V-Block-Adr.⟩/AB
```

Der TEST*FD-Befehl bewirkt bei Erreichen bestimmter Befehlszählerstände den Druck von Speicher- und/oder Indexbereichen. Der Versorgungsblock besteht aus Unterblöcken. Jeder Unterblock beginnt mit einer M-markierten Adreßkonstanten, die den Befehlszählerstand angibt. MN-Markierung bewirkt anschließend Ausschalten des Überwachers. Die folgenden Adreßkonstanten-Paare geben den zu druckenden Bereich an. N-Markierung bedeutet Indexbereich.

2.7. TEST*CR-Befehl

```
⟨TEST*CR-Befehl⟩  ::=  TEST*CR, ⟨V-Block-Adr.⟩/AB [M]
```

Der TEST*CR-Befehl weist den Überwacher an, die Registerstände zu drucken, falls ein Befehl den Adreßteil n oder m hat und diese Adresse in einem bestimmten Speicherbereich liegt. Der Versorgungsblock besteht aus Adreßkonstanten-Paaren, die den Speicherbereich angeben. Falls nur eine Adresse statt eines Adressenbereichs angegeben werden soll, so kann man statt des Adreßkonstanten-Paares eine M-markierte Adreßkonstante angeben.

2.8. TEST*XR-Befehl

```
⟨TEST*XR-Befehl⟩  ::=  TEST*XR, ⟨V-Block-Adr.⟩/AB [M]
```

Durch den TEST*XR-Befehl werden die Registerstände ausgedruckt, falls durch einen Indexbefehl (außer Doppelcodebefehlen) eine der im Versorgungsblock angegebenen Index- bzw. Speicherzellen angesprochen wird. Der Versorgungsblock besteht aus Adreßkonstanten, die Indexadressen oder Speicherhalbwortadressen darstellen. (Falls erste Adresse <256, dann werden alle Adressen als Indexadressen interpretiert; ansonsten Speicherhalbwortadressen.)

2.9. TEST*BR-Befehl

```
⟨TEST*BR-Befehl⟩  ::=  TEST*BR, ⟨V-Block-Adr.⟩/AB [M]
```

Durch den TEST*BR-Befehl werden die Registerstände ausgedruckt, falls der Befehlscode im Versorgungsblock aufgeführt ist (bei Doppelcodebefehl wird nur der Zweitcode beachtet). Der Versorgungsblock besteht aus Befehlen mit Adresse Null.

2.10. TEST*SPRUNG-Befehl

```
⟨TEST*SPRUNG-Befehl⟩  ::=  TEST*SPRUNG
```

Durch den TEST*SPRUNG-Befehl werden die Registerstände gedruckt, falls ein unbedingter Sprungbefehl mit erfüllter Bedingung vorliegt.

2.11. TEST*4XR-Befehl

```
⟨TEST*4XR-Befehl⟩  ::=  TEST*4XR
```

Spricht ein Programm die gleichen Indexzellen sowohl mit Indexbefehlen als auch mit Speicherbefehlen an und sollen diese überwacht werden, so ist es nötig, daß der Überwacher die 4 Indexregister simuliert (mit Alterungsmechanismus).

Durch den TEST*4XR-Befehl werden die 4 Indexregister simuliert.
Der TEST*4XR-Befehl wirkt bis zum nächsten TEST*4XR-, TEST*AUS- oder TEST*EIN-Befehl.

Rückspeicherung der 4 simulierten Indexregister erfolgt beim Ausschalten des Überwachers, bei SSR, BCI, ZI und falls Indexzellen-Dump verlangt wird, bei TEST*SD und TEST*FD.

2.12. TEST*FORT-Befehl (Überwachung des SSR-Befehls)

```
⟨TEST*FØRT-Befehl⟩  ::=  TEST*FØRT
```

Soll ein SSR-Befehl überwacht werden, so wird vorher die gesamte Druckinformation, die der Überwacher angesammelt hat, ausgegeben. Der Überwacher läßt dann den SSR-Befehls ausführen. Der Programmlauf wird nach dem SSR entweder fortgesetzt mit dem auf den SSR statisch folgenden Befehl (in diesem Fall bleibt der Überwacher normal angeschaltet) oder der Programmlauf wird fortgesetzt gemäß irgendeiner Adresse (z.B. gemäß einer im Versorgungsblock angegebenen Fehleradresse): in diesem Fall ist der Überwacher ausgeschaltet. Das Ausschalten läßt sich dadurch rückgängig machen, daß man an der Stelle, wo nach dem SSR hier weitergemacht wird (oder auch einige Befehle später, solange die Registerinhalte erhalten bleiben und kein Alarm erzeugt wird) den Befehl TEST*FORT gibt.

Dies bewirkt, daß Überwachung und Druck nach dem TEST*FORT-Befehl fortgesetzt werden wie vor dem SSR.

2.13. TEST*SSATZ-Befehl

```
⟨TEST*SSATZ-Befehl⟩  ::=  TEST*SSATZ, ⟨Anf.-Adr. Startsatz⟩/AB
```

Der TEST*SSATZ-Befehl bewirkt, daß aus dem Startsatz des Operatorlaufs der Parameter für die Druckseitenschranke des Überwachers ausgewertet wird (Voreinstellung: Druckseitenschranke des Überwachers = 30).
Dies geschieht aber schon implizit bei TEST*EIN.

Ist die Seitenschranke erreicht, so wird der Überwacher ausgeschaltet.
Man kann andererseits die Adresse eines simulierten Startsatzes, in dem nur das Drittelwort mit der Relativadresse 6 relevant ist (gibt die Seitenschranke an), mitgeben.

2.14. TEST*DATEI-Befehl

```
⟨TEST*DATEI-Befehl⟩  ::=  TEST*DATEI, ⟨V-Block-Adr.⟩/AB
```

Der Versorgungsblock ist ein Oktadenstring, der einen Dateinamen gemäß Kommandosyntax enthält.
Ist bei Erreichen des TEST*DATEI-Befehls diese Datei nicht vorhanden, so wird sie kreiert. Alle folgenden Druckausgaben des Überwachers gehen in diese Datei.

3. Allgemeines zu den Textbefehlen

Gibt man bei den Versorgungsblockadressen (bzw. bei der Befehlszahl (beim TEST*SR)) eine Markierung M an, so wird als 1. Zeichen einer Registerzeile anstelle von L gedruckt:

F bei TEST*FR
C bei TEST*CR
X bei TEST*XR
B bei TEST*BR
S bei TEST*SR

Die Versorgungsblöcke dürfen (außer bei TEST*SD und TEST*BR) höchstens 100 Halbworte enthalten. (Bei TEST*FR, bzw. TEST*CR zählt eine M-markierte Adreßkonstante doppelt; bei TEST*FD zählt ein M-markierter Befehlszählerstand vierfach).

Die Versorgungsblöcke werden mit 'FFFFFF'/H abgeschlossen. Die Testbefehle dürfen nicht modifiziert werden.

Die Spezifikation B der Versorgungsadreßkonstanten ist nur dann angebracht, wenn die Testbefehle im B-Bereich liegen.

4. Kontrollereignisse

4.1. KE-Befehl

```
⟨KE-Befehl⟩  ::=  KE␣⟨Name⟩
```

Der KE-Befehl wird nur dann interpretiert, wenn im UEBERSETZE-Kommando VARIANTE = GS angegeben ist (sonst wird er überlesen).
Er wird praktisch wie ein Makro expandiert und ergibt die TAS-Sequenz

```
SUE  T&KE,
  N  ("⟨Name⟩"),
```

wobei nur die ersten 6 Zeichen genommen werden, wenn der Name länger als 6 Zeichen ist. Eine Benennung wird an den generierten Text oder die folgende IE weitergereicht. T&KE ist der Kontaktname der Kontrollereignisprozedur S&KEP; die Externdeklaration wird implizit vom Assembler gegeben.

Beim Ansprung meldet sich das Kontrollereignis (falls aktiv) unter dem auf der vorherigen Seite erwähnten Namen und erwartet die Eingabe von Anweisungen: Näheres siehe Kapitel 6 des Kommandohandbuchs.

MAKROBIBLIOTHEKSORGANISATION

1.	Organisation einer Bibliothek als Datei	1
2.	Der Begriff der "Verfügbarkeit eines Makros"	1
3.	Aufruf eines Makros	2
4.	Pseudobefehle zum Beginnen und Beenden der Verfügbarkeit von Makros	2
4.1.	Der HDEF-Befehl (Hole Definition)	3
4.2.	Der LDEF-Befehl und der LDEFAB-Befehl	3
5.	Kommandos für Manipulationen mit Makrobibliotheken	4

I

MAKROBIBLIOTHEKSORGANISATION

Makrodefinitionen können entweder im TAS-Quellprogramm stehen oder bei Bedarf aus einer Makrobibliothek geholt werden.

1. Organisation einer Bibliothek als Datei

Eine "Makrobibliothek" ist im wesentlichen eine Datei, in der eine Menge von Makrodefinitionen gespeichert ist. Vom Benutzer aus gesehen ist diese Menge ungeordnet.

Jeder Satz der Datei (außer dem Satz Nr. 1) enthält ein Makro im Sinne der TAS-Sprache. Der Satz Nr. 1 enthält ein (hier nicht weiter beschriebenes) Inhaltsverzeichnis, das für n Makros aus 1+6n Ganzworten besteht.

Die Makronamen müssen innerhalb der betreffenden Makrobibliothek eindeutig sein.

Die Makronamen dürfen als Sonderzeichen nur & aber nicht *enthalten.

2. Der Begriff der "Verfügbarkeit eines Makros"

Ein Makro ist an einem Punkt eines TAS-Programms entweder "verfügbar" oder nicht. Verfügbar wird es nur auf folgende Weise: entweder

a durch eine im TAS-Quellenprogramm stehende (mit DEF beginnende) Makrodefinition, ausgenommen durch eine Definition innerhalb einer anderen Makrodefinition, oder

b durch einen HDEF-Befehl (siehe Kapitel I.4.1) oder

c durch den Aufruf eines (verfügbaren) Makros, bei dessen Ausführung die Bedingungen a oder b in Erfüllung gehen.

Für die momentan verfügbaren Makros ist bekannt

- in welcher Reihenfolge sie verfügbar wurden,
- ob sie durch einen Sammeltransport (siehe Kapitel I. 4. 1) einer ganzen Makrobibliothek verfügbar wurden,
- gegebenenfalls von welcher Bibliothek der Sammeltransport ausging.

Die Verfügbarkeit kann nur durch LDEF-Befehle oder LDEFAB-Befehle (siehe Kapitel I. 4. 2) enden.

3. Aufruf eines Makros

Nur verfügbare Makros können aufgrufen werden. Falls mehrere Makros mit gleichen Namen verfügbar sind, ist das zuletzt verfügbar gewordene gemeint.

4. Pseudobefehle zum Beginnen und Beenden der Verfügbarkeit von Makros

Anwendungshinweis:

Mit den folgenden Pseudobefehlen hat man es in der Hand, die Verfügbarkeitsbereiche der Makros beliebig festzulegen. Will man extrem während des Übersetzens an Speicherplatz sparen, so wird man jedes Makro erst unmittelbar vor seinem Aufruf verfügbar machen und schon unmittelbar danach seine Verfügbarkeit wieder beenden. Steht hingegen genügend viel Speicherplatz zur Verfügung, so wird man alle gewünschten Bibliotheksmakros schon am Programmanfang verfügbar machen und auf Befehle zum Beenden ihrer Verfügbarkeit ganz verzichten.

4.1. Der HDEF-Befehl (Hole Defintion)

```
⟨HDEF-Befehl⟩ ::= HDEF [␣⟨Datenbasisname⟩] (⟨Dateiname⟩ [ ,⟨Makroname⟩]0-20)
```

Anmerkung zur Syntax:

Bekanntlich darf der Datenbasisname höchstens 6 Zeichen lang sein und der Dateiname höchstens 12 Zeichen. Der Dateiname darf nicht, wie in der Kommandosprache, eine Generations- oder Versionsnummer enthalten.

Bedeutung: Die angegebene Datei (die dem Betriebssystem bereits bekannt sein muß) wird zur Bearbeitung eröffnet. Danach werden, falls eine oder mehrere Makronamen angegeben sind, die entsprechenden Makros aus der Datei geholt und in der Reihenfolge ihrer Satznumerierung verfügbar gemacht, anderenfalls (durch "Sammeltransport") alle Makros der Datei (in der Reihenfolge ihrer Satznumerierung). Danach wird die Dateibearbeitung beendet.

I

4.2. Der LDEF-Befehl und der LDEFAB-Befehl (Lösche Definition bzw. Lösche Definition ab)

In TAS stehen zum Löschen von Makrodefinitionen folgende Pseudobefehle zur Verfügung:

```
{ LDEF [␣⟨Datenbasisname⟩] (⟨Dateiname⟩)          }
{ LDEFAB [␣⟨Datenbasisname⟩] (⟨Dateiname⟩)       }
{ LDEF␣⟨Makroname⟩                               }
```

Bedeutung der ungeklammerten Form mit LDEF:

Unter allen verfügbaren Makros mit dem angegebenen Makronamen (ungeachtet evtl. Sammeltransport-Vermerke) endet die Verfügbarkeit des zuletzt verfügbar gewordenen Makros.

Bedeutung der geklammerten Form mit LDEF:

Die Verfügbarkeit aller aus dem letzten Sammeltransport der angegebenen Makrobibliothek noch verfügbaren Makros wird beendet. Zugleich wird auch der Vermerk über den Sammeltransport gelöscht. (Im Extremfall ist die Verfügbarkeit von jedem Makro dieses Sammeltransportes schon vorher einzeln beendet worden, so daß nun nur noch der Sammeltransport-Vermerk zu löschen ist.)

Bedeutung der Form mit LDEFAB:

Die Wirkung gleicht der Wirkung des LDEF-Befehls; außerdem werden noch für alle später verfügbar gewordenen Makros die Verfügbarkeit beendet und die Sammeltransport-Vermerke gelöscht.

Anmerkungen:

- Hinsichtlich des Zeitaufwandes ist der LDEFAB-Befehl günstiger als der LDEF-Befehl.
- Obige Befehle löschen Makrodefinitionen nur im Kernspeicher, nicht auch in der Makrobibliothek.
- LDEF und LDEFAB in Makros können zu Fehlern führen.

5. Kommandos für Manipulationen mit Makrobibliotheken

Eine Makrobibliothek ist entweder in der Datenbasis &OEFDB vorhanden (z.B. R&RAHMEN, siehe Kapitel G), oder wird vom Benutzer in die Standarddatenbasis &STDDB eingeschleust bzw. aus der LFD angemeldet.

Zum Aufbauen, Ändern und Drucken einer Makrobibliothek sind spezielle Kommandos verfügbar (siehe Kommando-Handbuch unter MEINTRAGE usw.). Man beachte, daß die Dateinamen von Makrobibliotheken weder eine Generationsnummer noch eine Versionsnummer enthalten dürfen.

GROSSE BEFEHLSLISTE ZUM PROGRAMMIEREN VON OPERATOREN IN DER PROGRAMMIERSPRACHE TAS

Bedeutung der Spalten	2
Erklärung der Zeichen	3
Transportbefehle	4-6
Festpunkt-Arithmetik	7
Gleitpunkt-Arithmetik	8
Boolesche Operationen	9
Halbwort-Arithmetik	9
Teilwort-Arithmetik	9
Index-Arithmetik	10
Setzen und Löschen	10, 11
Sprünge	12, 13
Modifizieren	14
Ersetzen (und modifizieren)	15
Aufbereitung	16, 17
Tabelle durchsuchen	17
Zentralcode	19
Potenzen von 2	20
Konvertierungstafel	21
Interncode-Externcode	22
Internspezifikationen	23
Wortstruktur	24
Blockschaltbild	25
Alphabetische Liste der Befehle	27

IAS IR 440

Nov. 72

Bedeutung der Spalten

Bezeichnung	Code	adr	Wirkung	mod2	R	veränderliche Spalten	⟨M⟩	Alarm BÜ	Alarm TK	Werk	Takte	Int.	Bemerkungen
①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	⑫	⑬	⑭
Erläuterungen des Mnemocodes	EMC	n	Hauptwirkung	+	x		x	E≷	≠1	B R	22	AB	Spezielles

* ⑮
**
*** Fußnoten

① Bezeichnung: Hier steht die Befehlsbezeichnung. Die Buchstaben die zum Mnemocode führen, sind unterstrichen.

② Code: Externcode (mnemotechnische Abkürzung der Befehlsbezeichnung)

③ adr: Adressenteil des Befehls

④ Wirkung: Hier wird die hauptsächliche Wirkung des Befehls angegeben. In den veränderlichen Spalten ⑦ werden auch evtl. auftretende Nebenwirkungen angegeben.

⑤ mod2: Vom vorhergehenden Befehl kann ein Modifikator 2. Art (mod2) vorhanden sein. Es ist hier angegeben wie dieser Modifikator auf den beschriebenen Befehl einwirkt.

+ = mod2 wird während der Abrufphase zum Adressenteil des Befehls addiert. Anschließend wird mod2 gelöscht. adr := adr + mod2; mod2:= 0

sp = Der Befehl wird während der Ausführungsphase speziell modifiziert. Die Art der Modifizierung ist bei diesen Befehlen angegeben.

leer = Ist in der Spalte mod2 keine Eintragung, so wird der Befehl nicht modifiziert; mod2 wird in der Abrufphase gelöscht. mod2 := 0

⑥ R: Ein x in dieser Spalte zeigt an, daß der Befehl Zweitcode beim Registerbefehl R zugelassen

⑦ veränderliche Spalten: Diese Spalten sind je nach Wirkung des Befehls unterschiedlich benannt. Im allgemeinen definieren diese Spalten den Inhalt der angegebenen Register nach Ausführung des Befehls. Bei Spalten ohne Eintragung und Registern, die nicht aufgeführt sind, erfolgt keine Änderung des betreffenden Inhaltes.
Folgende Schreibweisen werden verwendet:
Bei allen Registern mit Typenkennung und bei den Speicherzellen wird vor einem Semikolon die Typenkennung angegeben (t_a; = Typenkennung der Speicherzelle n).
Steht nach dem Semikolon und vor einem Komma 0 bzw. v, so wird damit ausgesagt, daß das Register links mit Null oder vorzeichengleich aufgefüllt wird (t_a;v,⟨m⟩ = In dem Register, das in dieser Spalte angegeben ist, steht der Inhalt der Speicherzelle m, links mit Vorzeichen aufgefüllt; beim Transport wird die TK aus der Speicherzelle m mit in dieses Register gebracht).

Ist eine Klammer ⌣ in zwei Registerspalten eingetragen, so steht in beiden Registern der gleiche Wert.

⑧ ⟨M⟩: Für das Markenregister ist durch x angegeben, daß bei diesem Befehl das Markenbit berücksichtigt wird. Als Wirkung gehört dazu, daß beim Transport eines Zahlwortes in ein Register das erste Bit dem zweiten angeglichen wird. Ist das Zahlwort markiert, so wird außerdem das Markenregister M gesetzt.
$\langle M\rangle := \langle M\rangle \vee \langle n\rangle_1$ $\qquad \langle a\rangle_1 := \langle a\rangle_2$
a = beliebiges Register

⑨ BÜ-Alarm: Hier wird angegeben, unter welchen Bedingungen ein Bereichsüberschreitungs-Alarm gegeben wird (nur bei TK 0 und 1).
> = übergelaufen ≷ = über- oder untergelaufen

⑩ TK-Alarm: Diese Spalte gibt an, bei welcher Bedingung ein Typenkennungs-Alarm auftritt.

⑪ Werk: Es wird angegeben, ob der Befehl das Befehlswerk (B), das Rechenwerk (R) oder beide belegt.

⑫ Takte: Hier wird die Ausführungszeit in Takten angegeben. Es handelt sich zum Teil um Mittelwerte. Sind unter "Werk" Befehls- und Rechenwerk aufgeführt, so gilt die Zeit für beide Werke. Für das Rechenwerk kann in vielen Fällen eine kürzere Zeit benötigt werden, die aber dann ohne Vorteil ist. Ist unter Werk nur das Rechenwerk aufgeführt, dann können während der angegebenen Zeit parallel dazu alle Befehle ablaufen, die nur das Befehlswerk ansprechen.

Zu den angegebenen Zeiten kommen für einfache Zeitberechnungen nachfolgend diese Zeiten hinzu, die in der Abrufphase liegen und das Befehlswerk belegen:

	Takte (Mittelwert)
Befehlsabruf	8
belegt Rechenwerk	1
Modifizierung 1. Art	8
Modifizierung 2. Art (wenn bei "mod2: +")	8
bei Operand aus dem Speicher	8
bei erfüllter Sprungbed.	5

Für genauere Zeitberechnungen s. TR 440 Befehls-Lexikon.

⑬ Int.: Diese Spalte nennt den Interncode des Befehls in zwei Sedezimalen.

⑭ Bemerkungen: Hier sind spezielle Vermerke und Erläuterungen zum Befehl aufgeführt.

⑮ Fußnoten: Sternchen in den Spalten werden in den Fußnoten erläutert.

Erklärung der Zeichen

Bezeichnung der Register im Rechenwerk:

A	Akkumulator	48 Bits Information 2 Bits Typenkennung
Q	Quotientenregister	
D	Multiplikandenregister	
H	Hilfsregister	
Y	Schiftzähler	8 Bits
M	Markenregister	1 Bit
A,Q	doppeltlanges Register A und Q	
H,Q	doppeltlanges Register H und Q	

Bezeichnung der Register im Befehlswerk:

B	Bereitadressenregister	24 Bits
F	Befehlsfolgeregister	
X	Indexbasisregister	22 Bits
K	Merklichterregister	8 Bits
U	Unterprogrammregister	

Variable:

mod1	Modifikator erster Art	24-Bit-Größe im Register B oder in einem nicht adressierbaren Register
mod2	Modifikator zweiter Art	
op	Operationscode eines Befehls 8 Bits	
adr	Inhalt des Adressenteils eines Befehls (16 Bits auf 24 Bits erweitert)	
n	Speicheradresse eines Ganzwortes nur geradzahlige Adressen, ungeradzahlige werden um 1 vermindert.	16 Bits
m	Speicheradresse eines Halbwortes	
z	Zahl oder Operand	
i	Indexadresse	Indizes: L für links, R für rechts
p	Parameter	
s	Spezifikation	
	(s evtl. unterteilt in $s_1, s_2, \ldots$)	
c	Zweitcode (Code für den Zweitbefehl)	

Zeichen und ihre Bedeutung:

:=	Die links stehende Zielgröße ergibt sich aus der rechts stehenden Quellengröße Beispiel: ⟨A⟩ := ⟨n⟩ Der Inhalt von A ergibt sich aus dem Inhalt von n.
:=:	Links- und rechtsstehende Größen werden miteinander vertauscht. Beispiel: ⟨A⟩ :=: ⟨H⟩ Die Inhalte von A und H werden vertauscht.
t_x;	Typenkennung im Register x oder in der Speicherzelle x
1,1;	Typenkennung in beiden Registern eines doppeltlangen Registers
0,	der linke Teil des Registers ist mit Null aufgefüllt
v,	der linke Teil des Registers ist vorzeichengleich aufgefüllt
⟨ ⟩	Inhalt eines Registers, einer Speicher- oder Indexzelle Beispiel: ⟨A⟩ Inhalt des Registers A (einschließlich Typenkennung)
⟨ ⟩,⟨ ⟩	Inhalt von zwei getrennten Registern Beispiel: ⟨A⟩,⟨Q⟩ Inhalt von A und Q
⟨ , ⟩	Inhalt zweier Register, die zusammengefaßt sind Beispiel: ⟨A,Q⟩ Inhalt der zu einem doppelt langen Register vereinigten Register A und Q
| |	Betrag einer Größe, Beispiel: |⟨n⟩| Betrag vom Inhalt der Speicherzelle n

Indizes für Teile eines Wortes:

$\langle\ \rangle_v$	Vorzeichen vom Inhalt
$\langle\ \rangle_t$	Typenkennung vom Inhalt
$\langle\ \rangle_m$	Marke vom Inhalt
$\langle A\rangle_1$	Bit 1 im Register A
$\langle A\rangle_{41-48}$	Bits 41 bis 48 im Register A
$\langle n\rangle_{9-24}$	Bits 9 bis 24 in der Speicherzelle (Drittelwort)
$\langle n\rangle_{1,2}$	Bits 1 und 2 in der Speicherzelle
	zur Zählung der Bits siehe Seite Wortstruktur

Logische Verknüpfungen:

∧ UND-Verknüpfung (Konjunktion)

a := b ∧ c	
0	0 0
0	0 L
0	L 0
L	L L

∨ ODER-Verknüpfung (Disjunktion)

a := b ∨ c	
0	0 0
L	0 L
L	L 0
L	L L

⫲ Antivalenz-Verknüpfung (exklusives ODER)

a := b ⫲ c	
0	0 0
L	0 L
L	L 0
0	L L

¬ Negation

a := ¬ b	
0	L
L	0

Transportbefehle

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨A⟩	⟨Q⟩	⟨H⟩	⟨D⟩	⟨Y⟩	⟨B⟩	⟨M⟩	Werk	Takte	Int.	Bemerkungen
Bringe (nach A)	B	n	⟨A⟩ := ⟨n⟩	+	x	t_n; ⟨n⟩						x	R	2	70	
Bringe unverändert	BU	n	⟨A⟩ := ⟨n⟩	+	x	t_n; ⟨n⟩							R	2	D3	unverändert bei jeder TK
Bringe und reserviere	BR	n	⟨H⟩ := ⟨A⟩ ⟨A⟩ := ⟨n⟩	+	x	t_n; ⟨n⟩		t_A;⟨A⟩				x	R	5	76	
Bringe negativ	BN	n	⟨A⟩ :=-⟨n⟩	+	x	t_n;-⟨n⟩						x	R	5	75	
Bringe negativ und reserviere	BNR	n	⟨H⟩ := ⟨A⟩ ⟨A⟩ :=-⟨n⟩	+	x	t_n;-⟨n⟩		t_A;⟨A⟩				x	R	5	77	
Bringe Betrag	BB	n	⟨A⟩ :=|⟨n⟩|	+	x	t_n;|⟨n⟩|						x	R	5	74	
Bringe nach Q	BQ	n	⟨Q⟩ := ⟨n⟩	+	x		t_n;⟨n⟩					x	R	3	72	
Bringe nach D	BD	n	⟨D⟩ := ⟨n⟩	+	x				t_n;⟨n⟩			x	R	3	71	
Bringe nach H	BH	n	⟨H⟩ := ⟨n⟩	+	x			t_n;⟨n⟩				x	R	3	73	
Bringe nach Q und bringe (nach A)	BQB	n	⟨A⟩ := ⟨n⟩ ⟨Q⟩ := ⟨n⟩	+	x	t_n; ⟨n⟩	t_n;⟨n⟩					x	R	2	DA	
Bringe zwei Wörter	BZ	n	⟨A⟩ := ⟨n⟩ ⟨Q⟩ := ⟨n+2⟩	+		t_n; ⟨n⟩	t_{n+2};⟨n+2⟩					x	B R	15	D9	
Bringe zwei Wörter negativ	BZN	n	⟨A⟩ :=-⟨n⟩ ⟨Q⟩ :=-⟨n+2⟩	+		t_n;-⟨n⟩	t_{n+2};-⟨n+2⟩					x	B R	15	D1	
Bringe Teilwort	BT	n	$\langle A\rangle_x$:=$\langle n\rangle_x$ falls $\langle Q\rangle_x$ = 0 $\langle A\rangle_x$:= 0 falls $\langle Q\rangle_x$ = L	+	x	t_n;... (s. Wirk.) ⟨A⟩ wird um p Stellen rechts geschiftet, 0 nachgezogen							R	7 +2p	F6	p: Anzahl der L, die rechtsbündig in Q stehen. x: 1,2,...,48
Bringe nächstes Zeichen	BNZ	i_L i_R	⟨A⟩ := 0, Zeichen gemäß a und b rechtsb. $\langle A\rangle_t$:= $\langle a\rangle_t$ b := b + 1 wenn b < d b := +0 } wenn b = d $\langle i_R\rangle$:= $\langle i_R\rangle$+2	sp		t_a; 0,... (s. Wirk.)			undefiniert		$\langle i_L\rangle_{alt}$		B R	41	E6	a: $\langle i_R\rangle$ + mod2 lauf. Adr. eines Wortes der Liste b: $\langle i_L\rangle_{17-24}$ lauf. Nummer eines Zeichens im Wort (0,1,...) d: (48/f)-1 max. Zeichennummer f: $\langle i_L\rangle_{9-12}$ Anzahl der Bits pro Zeich. (4,6,8 oder 12) andere Bits von $\langle i_L\rangle$ bedeutungslos
Bringe zwei Halbwörter	BZ2	m	⟨A⟩ := ⟨m,m+1⟩	+		t_m; ⟨m,m+1⟩							B R	10	D8	
Bringe Halbwort	B2	m	⟨A⟩ := ⟨m⟩	+	x	t_m; 0,⟨m⟩							R	9	6E	
Bringe Halbwort mit Vorzeichen	B2V	m	⟨A⟩ := ⟨m⟩	+	x	1; v,⟨m⟩							R	10	6F	
Bringe Halbwort mit Vorzeichen negativ	B2VN	m	⟨A⟩ :=-⟨m⟩	+	x	1;-(v,⟨m⟩)							R	12	67	
Bringe Drittelwort	B3	m	⟨A⟩ := $\langle m\rangle_{9-24}$	+	x	1; 0,$\langle m\rangle_{9-24}$							R	9	6C	
Bringe Drittelwort mit Vorzeichen	B3V	m	⟨A⟩ := $\langle m\rangle_{9-24}$	+	x	1; v,$\langle m\rangle_{9-24}$							R	10	6D	

Bringe aus Leitblock	BLEI	p	⟨A⟩ := (⟨BL⟩ · 2^8 + p)	+		t;(⟨BL⟩·2^8 + p)	B R	13	BE	p: +0...255 BL = Leitadressenregister
Bringe (und setze) Steuerbits (und Sperren)	BSS	s_L s_R	⟨A⟩ := Steuerbits falls a_9=L			2; Steuerbits falls a_9=L	B R	3	FB	s_L:Bits a_9 bis a_{16} s_R:Bits a_{17} bis a_{24} s_R ist ohne Bedeutung, muß aber angegeben werden.

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨n⟩ₐ	⟨n⟩	⟨n+2⟩	⟨A⟩	⟨M⟩	Alarm BÜ	Alarm TK	Werk	Takte	Int.	Bemerkungen
Bringe und lösche	BL	n	⟨A⟩ := ⟨n⟩ ⟨n⟩ := +0	+			0;0		t_a;⟨n⟩				B R	17	BO	unverändert bei jeder TK
Bringe und speichere	BC	n	⟨A⟩ :=: ⟨n⟩	+		⟨n⟩ₐ	t_A;⟨A⟩		t_a;⟨n⟩	x	≷		B R	13	A3	
Bringe und speichere Indexbasis	BCI	n	⟨X⟩ :=: ⟨n⟩₃₋₂₄ ⟨U⟩ :=: ⟨n⟩₄₁₋₄₈				3;0,⟨X⟩,0,⟨U⟩			⟨X⟩ ⟨n⟩₃₋₂₄ / ⟨U⟩ ⟨n⟩₄₁₋₄₈			B	17x +66	B6	x = Anzahl der zurückgespeicherten Indexreg.
Speichere	C	n	⟨n⟩ := ⟨A⟩	+		0	t_A;⟨A⟩				≷		B R	8	80	
Speichere unverändert	CU	n	⟨n⟩ := ⟨A⟩	+		⟨A⟩₁	t_A;⟨A⟩						B R	8	DO	unverändert bei jeder TK
Speichere und bringe Reserve	CR	n	⟨n⟩ := ⟨A⟩ ⟨A⟩ := ⟨H⟩	+		0	t_A;⟨A⟩		t_H;⟨H⟩		≷		B R	9	81	
Speichere negativ	CN	n	⟨n⟩ :=-⟨A⟩	+		0	t_A;-⟨A⟩				≷		B R	9	84	
Speichere Betrag	CB	n	⟨n⟩ :=|⟨A⟩|	+		0	t_A;|⟨A⟩|				≷		B R	9	85	
Speichere markiert	CMT	n	⟨n⟩ := ⟨A⟩	+		L	t_A;⟨A⟩				≷	2,3	B R	8	82	
Speichere mit Marke aus Register	CMR	n	⟨n⟩ := ⟨A⟩	+		⟨M⟩	t_A;⟨A⟩				≷	2,3	B R	8	83	
Speichere mit Marke aus Speicher	CMC	n	⟨n⟩ := ⟨A⟩	+		⟨n⟩ₐ	t_A;⟨A⟩				≷	2,3	B R	13	A2	
Speichere aus Q	CQ	n	⟨n⟩ := ⟨Q⟩	+		0	t_Q;⟨Q⟩				≷		B R	9	87	
Speichere aus D	CD	n	⟨n⟩ := ⟨D⟩	+		0	t_D;⟨D⟩				≷		B R	9	86	
Speichere aus H	CH	n	⟨n⟩ := ⟨H⟩	+		0	t_H;⟨H⟩				≷		B R	9	8F	
Speichere zwei Wörter	CZ	n	⟨n⟩ := ⟨A⟩ ⟨n+2⟩ := ⟨Q⟩	+		0	t_A;⟨A⟩	t_Q;⟨Q⟩			≷		B R	16	DB	
Speichere Teilwort	CT	n	⟨n⟩ₓ := ⟨A⟩ₓ	+		Wirkung: ⟨A⟩ um p Stellen links im Kreis geschiftet; ⟨n⟩ₓ := ⟨A⟩ₓ für ⟨Q⟩ₓ = 0; ⟨n⟩ₓ := ⟨n⟩ₓ für ⟨Q⟩ₓ = L; ⟨A⟩ im Kreis zurückgeschiftet		t_a;⟨n⟩ ⟨D⟩					B R	22 +2p	F7	p: Anzahl der L, die rechtsbündig im Reg. Q stehen x: 1,2,...,48
Speichere nächstes Zeichen	CNZ	i_L i_R	Voraussetzung: ⟨A⟩ = 0, f Bits rechte f Bits von ⟨A⟩ in a gemäß b eingesetzt ⟨A⟩ := ⟨a⟩ ⟨A⟩ₜ := ⟨a⟩ₜ b := b + 1 wenn b < d b := +0 ⟨i_R⟩ := ⟨i_R⟩+2 } wenn b = d	sp		⟨A⟩ t_a; ⟨a⟩	⟨n⟩	⟨B⟩ ⟨i_L⟩₈₁₁	⟨Y⟩ undef.	⟨Q⟩ abzusp. Zeichen aus A, gem.b geschiftet; andere Bits=0	⟨D⟩ undef.			B R	55	E7	a: ⟨i_R⟩ + mod2 lauf. Adr. eines Wortes der Liste b: ⟨i_L⟩₁₇₋₂₄ lauf. Nummer eines Zeichens im Wort (0,1,...) d: (48/f)-1 max. Zeichennummer f: ⟨i_L⟩₉₋₁₂ Anzahl der Bits pro Zeich. (4,6,8 oder 12) andere Bits ⟨i_L⟩ bedeutungslos
Speichere Halbwort	C2	m	⟨m⟩ := ⟨A⟩₂₅₋₄₈	+		⟨m⟩ ⟨A⟩₂₅₋₄₈							B R	15	AO	Das Halbwort wird unverändert eingesetzt
Speichere Drittelwort	C3	m	⟨m⟩₉₋₂₄ :=⟨A⟩₃₃₋₄₈	+		⟨m⟩₉₋₂₄ ⟨A⟩₃₃₋₄₈							B R	15	A1	Das Drittelwort wird unverändert eingesetzt

Transportbefehle (Fortsetzung)

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨B⟩	⟨i⟩	⟨i_L⟩	⟨i_R⟩	⟨s_1⟩	⟨m⟩	⟨K⟩	Werk	Takte	Int.	Bemerkungen
Registertausch	RT	s	⟨s_1⟩:=:⟨s_2⟩	+		⟨s_1⟩ und ⟨s_2⟩ werden vertauscht einschl. TK							R	4	97	s_1, s_2: A,Q,D oder H; $s_1 \neq s_2$
Index: Bringe	XB	i	⟨B⟩ := ⟨i⟩			⟨i⟩							B	1	08	Bits 9 - 11 = OLO
Index: Speichere	XC	i	⟨i⟩ := ⟨B⟩				⟨B⟩						B	3	18	Bit 9 = 0
Index: Speichere negativ	XCN	i	⟨i⟩ :=-⟨B⟩				-⟨B⟩						B	5		Bit 9 = L
Tausch-Transport in Indexzellen	TTX	i_L i_R	⟨B⟩ := ⟨i_L⟩ ⟨i_R⟩:=:⟨i_L⟩			⟨i_L⟩		⟨i_R⟩	⟨i_L⟩				B	11	OD	
Transport aus Indexzelle nach Indexzelle	TXX	i_L i_R	⟨i_L⟩:= ⟨i_R⟩ ⟨B⟩ := ⟨i_R⟩			⟨i_R⟩		⟨i_R⟩					B	4	OC	
Transport aus Indexzelle nach Rechenwerk	TXR	s i	⟨s_1⟩:=±⟨i⟩ ⟨B⟩ :=±⟨i⟩			±⟨i⟩				1;v,±⟨i⟩			B R	5	8C	s_1: A,Q,D und H s_2: leer: + N : - } statt ±
Transport aus Rechenwerk nach Indexzelle	TRX	s i	⟨i⟩ :=±⟨s_1⟩$_{25-48}$ ⟨B⟩ :=±⟨s_1⟩$_{25-48}$			±⟨s_1⟩$_{25-48}$							B R	6	9C	s_1: A,Q,D,H oder leer s_2: leer: + N : - } statt ±
Transport aus Speicher nach B	TCB	m	⟨B⟩ := ⟨m⟩	+	x	⟨m⟩							B	2	39	
Transport aus B nach Speicher	TBC	m	⟨m⟩ := ⟨B⟩								⟨B⟩		B	7	07	
Bringe und speichere Merklichter	BCL	m	⟨m⟩$_{17-24}$:=: ⟨K⟩								⟨K⟩	⟨m⟩$_{17-24}$	B	18	06	⟨m⟩$_{1-16}$ bleiben erhalten

						Voraussetzung			
Wortgruppentransport vorwärts	WTV	i_L i_R	⟨⟨i_L⟩+2k⟩ := ⟨⟨i_R⟩+2k⟩ ⟨B⟩ := ⟨i_L⟩+2(⟨B⟩-1)			⟨B⟩ = Länge der Wortgruppen in Ganzwörtern = a ⟨i_R⟩= niederste Adresse des Quellenbereichs ⟨i_L⟩= niederste Adresse des Zielbereichs	B	21a +10	22	k; 0, 1, 2, 3,..., ⟨B-1⟩ a: Anzahl der Ganzwörter
Wortgruppentransport rückwärts	WTR	i_L i_R	⟨⟨i_L⟩-2k⟩ := ⟨⟨i_R⟩-2k⟩ ⟨B⟩ := ⟨i_L⟩-2(⟨B⟩-1)			⟨B⟩ = Länge der Wortgruppen in Ganzwörtern = a ⟨i_R⟩= höchste Adresse des Quellenbereichs ⟨i_L⟩= höchste Adresse des Zielbereichs	B	21a +10	23	

						Wirkung				
Bequemes Bringen aller Register	QBR	n	Die Register werden unverändert gebracht			⟨T⟩ := ⟨n⟩$_{1-24}$ ⟨H⟩ := t_{n-2}; ⟨n-2⟩ ⟨D⟩ := t_{n-4}; ⟨n-4⟩ ⟨Q⟩ := t_{n-6}; ⟨n-6⟩ ⟨A⟩ := t_{n-8}; ⟨n-8⟩ ⟨M⟩ := L, falls ⟨n-10⟩$_t$=1 oder 3 ⟨M⟩ := 0, falls ⟨n-10⟩$_t$=0 oder 2 ⟨B⟩ := ⟨n-10⟩$_{1-24}$ ⟨K⟩ := ⟨n-10⟩$_{25-32}$ ⟨Y⟩ := ⟨n-10⟩$_{33-40}$ ⟨U⟩ := ⟨n-10⟩$_{41-48}$	B R	84	FE	T: Prüfregister (nicht zugreifbares Register)
Bequemes Speichern aller Register	QCR	n	Die Register werden unverändert abgespeichert			⟨n⟩$_t$:= 2 + ⟨M⟩ ⟨n⟩$_{1-24}$:= ⟨B⟩ ⟨n⟩$_{25-32}$:= ⟨K⟩ ⟨n⟩$_{33-40}$:= ⟨Y⟩ ⟨n⟩$_{41-48}$:= ⟨U⟩ ⟨n+2⟩ := t_A;⟨A⟩ ⟨n+4⟩ := t_Q;⟨Q⟩ ⟨n+6⟩ := t_D;⟨D⟩ ⟨n+8⟩ := t_H;⟨H⟩ ⟨n+10⟩:= 2; ⟨T⟩,0	B R	55	FF	T: Prüfregister (nicht zugreifbares Register)

						Spezifikationen				
Zeichenkettenverarbeitung Version Bringen Version Speichern	ZK	s i	⟨A⟩bzw.⟨A,Q⟩:= p Oktaden gem.a;ggf.verkürzen zu Tetraden ⟨a⟩:=⟨A⟩bzw.⟨A,Q⟩p Okt. ggf.verlängern aus Tetraden			s_1:leer/B = Speichern/Bringen s_2:leer/I = Oktadenadr.nicht erhöhen/erhöhen um p (Inkrement) s_3:leer/V = nicht verändern/verändern (Oktaden—Tetraden) s_4:leer/R = Oktadenadresse in Indexzelle/oder Register s_5: p = Anzahl der Oktaden (1...12) i:Indexadresse wenn s_4 = leer; Reg.D,H od.B wenn s_4 = R	B R	min. 122	FC	a=⟨i⟩wenn s_4=leer a=⟨D⟩$_{25-48}$wenn i=D a=⟨H⟩$_{25-48}$wenn i=H a=⟨B⟩ wenn i=B } und s_4=R a: Oktadenadresse

					⟨A⟩	⟨D⟩	⟨H⟩	⟨Q⟩	⟨B⟩	⟨M⟩					
Transportie. Oktaden wenn $\langle H\rangle_t = 3$ wenn $\langle H\rangle_t = 2$	TOK	z	⟨a+x⟩:=⟨q+x, Quellgeb.z Okt. (q+6)≥a ⟨a+x⟩:=⟨q+y⟩Quellgeb.6 Okt. i.ein Ganzwort q=3n+0		3;Inhalt d.letzten Ganzwortes im Zielgebiet 3;Inhalt d.letzten Ganzwortes im Zielgebiet		2;⟨a+z⟩,q	3;0	letzte Halbw. adr.i. Zielg.	x	B	R	min. 191 min. 209	FD	z: 1...65535 a: $\langle H\rangle_{1-24}$=Okt.adr.Zielgebiet q: $\langle H\rangle_{25-48}$=Okt.adr.Quellgebiet x: 0,1,2,...,z-1 y: 0,1,2,3,4,5, 0, 1, 2, ...

Festkomma-Arithmetik

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨A⟩	⟨Q⟩	⟨D⟩	⟨n⟩	⟨Y⟩	⟨M⟩	Alarm BÜ	Alarm TK	Werk	Takte	Int.	Bemerkungen
Addiere	A	n	⟨A⟩ := ⟨A⟩ + ⟨n⟩	+	x	t_{max};⟨A⟩+⟨n⟩		t_n;⟨n⟩			x	≷		R	8	42	
Addiere Betrag	AB	n	⟨A⟩ := ⟨A⟩ +|⟨n⟩|	+	x	t_{max};⟨A⟩+|⟨n⟩|		t_n;⟨n⟩			x	E≷		R	8	40	
Addiere im Speicher	AC	n	⟨n⟩ := ⟨n⟩ + ⟨A⟩	+				t_{max};⟨n⟩+⟨A⟩	t_{max};⟨n⟩+⟨A⟩		x	≷		B R	17	43	
Subtrahiere	SB	n	⟨A⟩ := ⟨A⟩ - ⟨n⟩	+	x	t_{max};⟨A⟩-⟨n⟩		t_n;⟨n⟩			x	≷		R	8	46	
Subtrahiere Betrag	SBB	n	⟨A⟩ := ⟨A⟩ -|⟨n⟩|	+	x	t_{max};⟨A⟩-|⟨n⟩|		t_n;⟨n⟩			x	≷		R	8	41	
Subtrahiere im Speicher	SBC	n	⟨n⟩ := ⟨n⟩ - ⟨A⟩	+				t_{max};⟨n⟩-⟨A⟩	t_{max};⟨n⟩-⟨A⟩		x	≷		B R	17	47	
Subtrahiere invers	SBI	n	⟨A⟩ := ⟨n⟩ - ⟨A⟩	+	x	t_{max};⟨n⟩-⟨A⟩		t_n;⟨n⟩			x	≷		R	8	44	
Subtrahiere von D	SBD	n	⟨A⟩ := ⟨D⟩ - ⟨n⟩	+	x	t_{max};⟨D⟩-⟨n⟩					x	≷		R	11	45	
Multipliziere mit Rundung	MLR	n	⟨A⟩ := ⟨A⟩ • ⟨n⟩	+	x	1;(⟨A⟩•⟨n⟩)*	1;+0	t_n;⟨n⟩		+0	x		≠1	R	57	55	
Multipliziere negativ mit Rundung	MNR	n	⟨A⟩ :=-⟨A⟩ • ⟨n⟩	+	x	1;(-⟨A⟩•⟨n⟩)*	1;+0	t_n;⟨n⟩		+0	x		≠1	R	57	59	
Multipliziere akkumulierend mit Rundung	MAR	n	⟨A⟩ := ⟨A⟩ • ⟨n⟩ + ⟨H⟩	+	x	1;(⟨A⟩•⟨n⟩)*+⟨H⟩	1;+0	t_n;⟨n⟩		+0	x	≷	≠1	R	67	57	
Multipliziere akkumulierend negativ m. Rund.	MANR	n	⟨A⟩ :=-⟨A⟩ • ⟨n⟩ + ⟨H⟩	+	x	1;(-⟨A⟩•⟨n⟩)*+⟨H⟩	1;+0	t_n;⟨n⟩		+0	x	≷	≠1	R	67	5B	
Dividiere	DV	n	⟨A⟩ := ⟨A⟩ : ⟨n⟩	+	x	1;⟨A⟩:⟨n⟩	1;Rest•2^{48}	1;+0		+0	x	**	≠1	R	220	60	Operanden, Quotient und Rest sind als echte Brüche betrachtet (Komma links)
Dividiere invers	DVI	n	⟨A⟩ := ⟨n⟩ : ⟨A⟩	+	x	1;⟨n⟩:⟨A⟩	1;Rest•2^{48}	1;+0		+0	x	**	≠1	R	222	62	
Dividiere doppelt lang	DVD	n	⟨A⟩ := ⟨A,Q⟩ : ⟨n⟩	+	x	1;⟨A,Q⟩:⟨n⟩	1;Rest•2^{48}	1;+0		+0	x	**	≠1	R	228	61	
						⟨A,Q⟩											
Addiere in AQ	AQ	n	⟨A,Q⟩ := ⟨A,Q⟩ + ⟨n⟩	+	x	1,1;⟨A,Q⟩+⟨n⟩		t_n;⟨n⟩			x	≷	≠1	R	17	7E	Operand und Ergebnis: $\langle A\rangle_v$ kann ≠ $\langle Q\rangle_v$
Subtrahiere in AQ	SBQ	n	⟨A,Q⟩ := ⟨A,Q⟩ - ⟨n⟩	+	x	1,1;⟨A,Q⟩-⟨n⟩		t_n;⟨n⟩			x	≷	≠1	R	17	7F	
Multipliziere	ML	n	⟨A,Q⟩ := ⟨A⟩ • ⟨n⟩	+	x	1,1;⟨A⟩•⟨n⟩		t_n;⟨n⟩		+0	x		≠1	R	55	54	Ergebnis: $\langle A\rangle_v = \langle Q\rangle_v$
Multipliziere negativ	MLN	n	⟨A,Q⟩ :=-⟨A⟩ • ⟨n⟩	+	x	1,1;-⟨A⟩•⟨n⟩		t_n;⟨n⟩		+0	x		≠1	R	55	58	
Multipliziere akkumulierend	MLA	n	⟨A,Q⟩ := ⟨A⟩ • ⟨n⟩ + ⟨H,Q⟩	+	x	1,1;⟨A⟩•⟨n⟩+⟨H,Q⟩		t_n;⟨n⟩		+0	x	≷	≠1	R	66	56	$\langle H\rangle_t$ und $\langle Q\rangle_t$ ohne Bedeutung für die Ausführung des Befehls
Multipliziere akkumulierend negativ	MAN	n	⟨A,Q⟩ :=-⟨A⟩ • ⟨n⟩ + ⟨H,Q⟩	+	x	1,1;-⟨A⟩•⟨n⟩+⟨H,Q⟩		t_n;⟨n⟩		+0	x	≷	≠1	R	66	5A	Operand: $\langle H\rangle_v$ kann ≠ $\langle Q\rangle_v$ Ergeb. : $\langle A\rangle_v$ kann ≠ $\langle Q\rangle_v$
						⟨A⟩											
Addiere Adressenteil	AA	z	⟨A⟩ := ⟨A⟩ + z	+		TK = 0: 0;⟨A⟩•$16^{\pm z}$ TK = 1: 1;⟨A⟩±v,z TK = 2: 2;⟨A⟩$_{1-24}$,(⟨A⟩$_{25-48}$±z) TK = 3: 3;⟨A⟩±v,z	1;0,z 1;v,z 1;0,z 1;v,z				> ≷			R	13	98	TK = 0: |z| ≤ 2^7 TK = 1: |z| ≤ 2^{23} TK = 2: z ≤ 2^{16} TK = 3: |z| < 2^{23} nach Modifiz.
Subtrahiere Adressenteil	SBA	z	⟨A⟩ := ⟨A⟩ - z	+										R	13	99	

* gerundet ** Voraussetzung: bei DV, DVD: $|\langle A\rangle| < |\langle n\rangle|$
bei DVI : $|\langle n\rangle| < |\langle A\rangle|$
im anderen Fall: Gleitkommaergebnis, BÜ-Alarm

Gleitkomma-Arithmetik

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨A⟩	⟨n⟩	⟨D⟩	⟨Q⟩	⟨Y⟩	⟨M⟩	Alarm BÜ	Alarm TK	Werk	Takte	Int.	Bemerkungen
Gleitkomma addiere	GA	n	⟨A⟩ := ⟨A⟩ + ⟨n⟩	+	x	0;⟨A⟩+⟨n⟩*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	28	4B	
Gleitkomma addiere Betrag	GAB	n	⟨A⟩ := ⟨A⟩ +|⟨n⟩|	+	x	0;⟨A⟩+|⟨n⟩|*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	28	52	
Gleitkomma addiere im Speicher	GAC	n	⟨n⟩ := ⟨A⟩ + ⟨n⟩	+			0;⟨A⟩+⟨n⟩*	0;⟨A⟩+⟨n⟩*	0;+0	**	x	>	≠0	B R	37	4A	
Gleitkomma subtrahiere	GSB	n	⟨A⟩ := ⟨A⟩ - ⟨n⟩	+	x	0;⟨A⟩-⟨n⟩*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	28	4F	
Gleitkomma subtrahiere Betrag	GSBB	n	⟨A⟩ := ⟨A⟩ -|⟨n⟩|	+	x	0;⟨A⟩-|⟨n⟩|*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	28	53	
Gleitkomma subtrahiere im Speicher	GSBC	n	⟨n⟩ := ⟨n⟩ - ⟨A⟩	+			0;⟨n⟩-⟨A⟩*	0;⟨n⟩-⟨A⟩*	0;+0	**	x	>	≠0	B R	37	4E	
Gleitkomma subtrahiere invers	GSBI	n	⟨A⟩ := ⟨n⟩ - ⟨A⟩	+	x	0;⟨n⟩-⟨A⟩*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	28	48	
Gleitkomma subtrahiere von D	GSBD	n	⟨A⟩ := ⟨D⟩ - ⟨n⟩	+	x	0;⟨D⟩-⟨n⟩*			0;+0	**	x	>	≠0	R	30	4C	
Gleitkomma multipliziere	GML	n	⟨A⟩ := ⟨A⟩ · ⟨n⟩	+	x	0;⟨A⟩·⟨n⟩*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	54	5E	
Gleitkomma multipliziere negativ	GMLN	n	⟨A⟩ :=-⟨A⟩ · ⟨n⟩	+	x	0;-⟨A⟩·⟨n⟩*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	54	5C	
Gleitkomma multipliziere akkumulierend	GMLA	n	⟨A⟩ := ⟨A⟩ · ⟨n⟩ + ⟨H⟩	+	x	0;⟨A⟩·⟨n⟩+⟨H⟩*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	97	5F	
Gleitkomma multipliziere akkum. negativ	GMAN	n	⟨A⟩ :=-⟨A⟩ · ⟨n⟩ + ⟨H⟩	+	x	0;-⟨A⟩·⟨n⟩+⟨H⟩*		t_a;⟨n⟩	0;+0	**	x	>	≠0	R	97	5D	
Gleitkomma dividiere	GDV	n	⟨A⟩ := ⟨A⟩ : ⟨n⟩	+	x	0;⟨A⟩:⟨n⟩*		0;+0	0;+0	**	x	>	≠0	R	213	64	
Gleitkomma dividiere invers	GDVI	n	⟨A⟩ := ⟨n⟩ : ⟨A⟩	+	x	0;⟨n⟩:⟨A⟩*		0;+0	0;+0	**	x	>	≠0	R	216	66	
Addiere unnormalisiert	AU	n	⟨A⟩ := ⟨A⟩ + ⟨n⟩	+	x	0;⟨A⟩+⟨n⟩		t_a;⟨n⟩	0;+0	+0	x	>	≠0	R	26	49	
Subtrahiere unnormalisiert	SBU	n	⟨A⟩ := ⟨A⟩ - ⟨n⟩	+	x	0;⟨A⟩-⟨n⟩		t_a;⟨n⟩	0;+0	+0	x	>	≠0	R	26	4D	
Bilde reziproken Wert	REZ	n	⟨A⟩ := 1 : ⟨n⟩	+	x	0;1:⟨n⟩*		0;+0	0;+0	**	x	>	≠0	R	213	65	
						⟨A,Q⟩			⟨H⟩								
Doppelte Genauigkeit: Addiere	DA	n	⟨A,Q⟩ := ⟨A,Q⟩+⟨n,n+2⟩			0,1;⟨A,Q⟩+⟨n,n+2⟩*		1;+0	1;+0	**	x	>	2,3	BR	106	F0	⟨A⟩$_t$ = ⟨n⟩$_t$ = 0 ⟨Q⟩$_t$ = ⟨n+2⟩$_t$ = 1
Doppelte Genauigkeit: Subtrahiere	DSB	n	⟨A,Q⟩ := ⟨A,Q⟩-⟨n,n+2⟩			0,1;⟨A,Q⟩-⟨n,n+2⟩*		1;+0	1;+0	**	x	>	2,3	BR	106	F1	
Doppelte Genauigkeit: Multipliziere	DML	n	⟨A,Q⟩ := ⟨A,Q⟩·⟨n,n+2⟩			0,1;⟨A,Q⟩·⟨n,n+2⟩*		1;+0	1;+0	**	x	>	2,3	BR	243	F2	
Gleitkomma multipliz. auf doppelte Genauigk.	MLD	n	⟨A,Q⟩ := ⟨A⟩ · ⟨n⟩		x	0,1;⟨A⟩·⟨n⟩*		1;+0	1;+0	**	x	>	≠0	BR	64	F3	
						⟨A⟩											
Addiere Adressenteil (TK = 0)	AA	z	⟨A⟩ := ⟨A⟩ · 16^z	+		0;⟨A⟩·16^{+z}		1;0,z				>		R	13	98	nur bei ⟨A⟩$_t$ = 0; bei TK ≠ 0 siehe Festkomma-Arithmetik. |z| < 2^7 nach Modifizierung
Subtrahiere Adressenteil (TK = 0)	SBA	z	⟨A⟩ := ⟨A⟩ · 16^{-z}	+		0;⟨A⟩·16^{-z}		1;0,z				>		R	13	99	

* normalisiert und gerundet

** Anzahl der Binärstellen um die das Ergebnis normalisiert wurde.
Falls Ergebnis = ±0 oder Exponentenunterlauf: ⟨Y⟩ := +0

Index-Arithmetik

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨B⟩	⟨i⟩	⟨i_R⟩		Werk	Takte	Int.	Bemerkungen
Erhöhe B um Speicher	HBC	m	⟨B⟩ := ⟨B⟩ + ⟨m⟩		x	⟨B⟩+⟨m⟩				B	6,5	3C	
Vermindere B um Speicher	VBC	m	⟨B⟩ := ⟨B⟩ - ⟨m⟩		x	⟨B⟩-⟨m⟩				B	8,5	15	
Erhöhe B um Adressenteil	HBA	z	⟨B⟩ := ⟨B⟩ + z			⟨B⟩+ z				B	7,5	11	z: 0...65 535
Vermindere B um Adressenteil	VBA	z	⟨B⟩ := ⟨B⟩ - z			⟨B⟩- z				B	8,5	13	
Erhöhe B um Parameter mal Indexzelle	HBPX	p i	⟨B⟩ := ⟨B⟩ + p · ⟨i⟩			⟨B⟩+ p·⟨i⟩				B	5,5|p|+3	0F	p: ±1...±15
Erhöhe Indexzelle um Parameter	HXP	p i	⟨B⟩ := ⟨i⟩ + p ⟨i⟩ := ⟨i⟩ + p			⟨i⟩ + p	⟨i⟩ + p			B	10,5	2C	p: ±0...±127
Erhöhe Indexzelle um Indexzelle	HXX	i_L i_R	⟨B⟩ := ⟨i_R⟩ + ⟨i_L⟩ ⟨i_R⟩:= ⟨i_R⟩ + ⟨i_L⟩			⟨i_R⟩+⟨i_L⟩		⟨i_R⟩+⟨i_L⟩		B	14,5	2E	
Vermindere Indexzelle um Indexzelle	VXX	i_L i_R	⟨B⟩ := ⟨i_R⟩ - ⟨i_L⟩ ⟨i_R⟩:= ⟨i_R⟩ - ⟨i_L⟩			⟨i_R⟩-⟨i_L⟩		⟨i_R⟩-⟨i_L⟩		B	14,5	2F	
Register und Indexzelle	RX	s i	⟨B⟩ := ⟨i⟩ ± ⟨s_1⟩ Falls s_3 = C: ⟨i⟩ := ⟨i⟩ ± ⟨s_1⟩			⟨i⟩±⟨s_1⟩ ⟨i⟩±⟨s_1⟩	nur falls s_3 = C ⟨i⟩±⟨s_1⟩			B R	12	* 8D	s_1: A,Q,D,H (rechtes Halbwort) oder B s_2: leer = positiv + / N = negativ - } statt ± s_3: leer = nicht zurückspeich. / C = zurückspeichern

*siehe Internspezifikation auf Seite 23

Setzen und Löschen

						⟨A⟩	⟨H⟩	⟨s_2⟩					
Bringe Adressenteil	BA	z	⟨A⟩ := z	+		1;v,z				R	2	8E	z: 0...65535
Bringe Adressenteil und reserviere	BAR	z	⟨H⟩ := ⟨A⟩ ⟨A⟩ := z	+		1;v,z	t_A;⟨A⟩			R	3	DC	
Bringe Adressenteil negativ	BAN	z	⟨A⟩ := -z	+		1;-(v,z)				R	5	DF	
Bringe Adressenteil negativ und reserviere	BANR	z	⟨H⟩ := ⟨A⟩ ⟨A⟩ := -z	+		1;-(v,z)	t_A;⟨A⟩			R	5	DD	
Lösche Register	LR	s	⟨s_2⟩ := +0 ⟨s_2⟩$_t$:= s_1	+				s_1;+0		R	3	9A	s_1: 0,1,2 oder 3 (TK) s_2: A,Q,D und H
Lösche in A	LA	s	⟨A⟩$_s$:= 0 ⟨M⟩ := 0 nur bei s = M	+		Wirkung s = F: ⟨A⟩$_{1-40}$:= 0 ≙ Mantissenteil s = 2: ⟨A⟩$_{1-24}$:= 0 ≙ linkes Halbwort s = E: ⟨A⟩$_{41-48}$:= 0 ≙ Exponententeil s = 3: ⟨A⟩$_{33-48}$:= 0 ≙ rechtes Drittel s = V: ⟨A⟩$_{1-2}$:= 0 ≙ Vorzeichenstellen s = M: ⟨M⟩ := 0 ≙ Markenregister s = H: ⟨A⟩$_{1-42}$:= 0 ≙ ohne rechte Hexade s = T: ⟨A⟩$_{1-44}$:= 0 ≙ ohne rechte Tetrade			Die Spezifikationen F bis M können kombiniert verwendet werden H und T nur einzeln od. mit M erlaubt	R	4	8B	

Boolesche Operationen

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨A⟩	⟨D⟩		Alarm ⟨M⟩	BÜ	Werk	Takte	Int.	Bemerkungen
VEL	VEL	n	⟨A⟩ := ⟨A⟩ ∨ ⟨n⟩	+	x	t_{max};⟨A⟩∨⟨n⟩	t_n;⟨n⟩		x		R	5	68	
AUT	AUT	n	⟨A⟩ := ⟨A⟩ ≢ ⟨n⟩	+	x	t_{max};⟨A⟩≢⟨n⟩	t_n;⟨n⟩		x		R	5	69	
ET	ET	n	⟨A⟩ := ⟨A⟩ ∧ ⟨n⟩	+	x	t_{max};⟨A⟩∧⟨n⟩	t_n;⟨n⟩		x		R	5	6A	
Setze zusammen	ZUS	n	$\langle A\rangle_x$:= $\langle A\rangle_x$: für $\langle H\rangle_x$ = 0 $\langle A\rangle_x$:= $\langle n\rangle_x$: für $\langle H\rangle_x$ = L	+	x	t_{max}(A,n) ...; (s. Wirkung)	t_n;⟨n⟩		x		R	5	6B	
VEL Adressenteil	VLA	z	⟨A⟩ := ⟨H⟩ ∨ z	+		t_H;⟨H⟩∨(0,z)	1;0,z				R	8	88	z: 0...65535
AUT Adressenteil	ATA	z	⟨A⟩ := ⟨H⟩ ≢ z	+		t_H;⟨H⟩≢(0,z)	1;0,z				R	8	89	z: 0...65535
ET Adressenteil	ETA	z	⟨A⟩ := ⟨H⟩ ∧ z	+		t_H;⟨H⟩∧(0,z)	1;0,z				R	8	8A	z: 0...65535

Halbwort-Arithmetik

						⟨A⟩	⟨Q⟩	⟨D⟩	⟨Y⟩							
Addiere Halbwort	A2	m	⟨A⟩ := $\langle A\rangle_{25-48}$ + ⟨m⟩	+	x	t_A;v,(($\langle A\rangle_{25-48}$+⟨m⟩))		t_A;+0 .					≥	R	14	7C	
Subtrahiere Halbwort	SB2	m	⟨A⟩ := $\langle A\rangle_{25-48}$ - ⟨m⟩	+	x	t_A;v,(($\langle A\rangle_{25-48}$-⟨m⟩))		t_A;+0					≥	R	14	7D	
Multipliziere Halbwort	M2	m	⟨A⟩ := ⟨A⟩ · ⟨m⟩	+	x	t_A;⟨A⟩·⟨m⟩	t_A;+0	t_A;+0	+0				≥	R	49,5	7A	beim Ergebnis entfallen die linken 24 Bits
Multipliziere Halbwort negativ	M2N	m	⟨A⟩ :=-⟨A⟩ · ⟨m⟩	+	x	t_A;-⟨A⟩·⟨m⟩	t_A;+0	t_A;+0	+0				≥	R	49,5	78	
Multipliziere Halbwort mit Rundung	M2R	m	⟨A⟩ := ⟨A⟩ · ⟨m⟩ gerundet	+	x	t_A;(⟨A⟩·⟨m⟩) gerundet	t_A;+0	t_A;+0	+0					R	36,5	7B	Beim Ergebnis entfallen die rechten 24 Bits und es wird gerundet
Multipliziere Halbwort negativ mit Rundung	M2NR	m	⟨A⟩ :=-⟨A⟩ · ⟨m⟩ gerundet	+	x	t_A;(-⟨A⟩·⟨m⟩) gerundet	t_A;+0	t_A;+0	+0					R	36,5	79	

Teilwort-Arithmetik

						⟨A⟩	⟨D⟩					
Addiere Teilwort	AT	n	⟨A⟩ := $\langle A\rangle_x$ + $\langle n\rangle_x$	+	x	Hilfsgröße qr := um p Stellen rechts im Kreis geschifteter ⟨Q⟩ Hilfsgröße nr := um p Stellen rechts geschifteter ⟨n⟩ Das Nullfeld von qr schneidet in nr und ⟨A⟩ Teilwörter aus; diese werden addiert bzw. subtrahiert. Die anderen Stellen werden auf 0 gelöscht. t_A;Ergebnis	t_Q;⟨Q⟩	>	R	21+2p	F4	p = Anzahl der rechts in Q anstehenden L-Bits Falls ⟨Q⟩ = LL...L dann p = 0
Subtrahiere Teilwort	SBT	n	⟨A⟩ := $\langle A\rangle_x$ - $\langle n\rangle_x$	+	x			<			F5	Die Zahlen werden als positive, vorzeichenlose, ganze Festkommazahlen aufgefaßt.

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨B⟩	⟨i⟩	⟨U⟩		Werk	Takte	Int.	Bemerkungen
Index: Bringe Adressenteil	XBA	z	⟨B⟩ := z			z				B	1	01	z: 0...65535
Index: Bringe Adressenteil negativ	XBAN	z	⟨B⟩ := -z			-z				B	2	19	
Setze Index	ZX	p i	⟨i⟩ := p ⟨B⟩ := p			v,p	v,p			B	6	1A	p: ±0...±127
Setze Unterprogrammregister	ZU	i	⟨U⟩ := i	+				i		B	1	3E	

Bezeichnung	Code	adr	Wirkung	mod2	R	$\langle n\rangle_t$	$\langle s_2\rangle_t$	⟨M⟩		Werk	Takte	Int.	Bemerkungen
Setze Typenkennung im Register	ZTR	s	$\langle s_2\rangle_t := s_1$ ⟨M⟩ := L, falls s_3 = M	+			s_1	L, falls s_3 = M		R	5	92	s_1: 0,1,2,3 oder leer s_2: A,Q,D,H oder leer s_3: leer oder M
Setze Typenkennung 0	ZT0	n	$\langle n\rangle_t$:= 0	+		0				B	13	C8	
Setze Typenkennung 1	ZT1	n	$\langle n\rangle_t$:= 1	+		1				B	13	C9	
Setze Typenkennung 2	ZT2	n	$\langle n\rangle_t$:= 2	+		2				B	13	CA	
Setze Typenkennung 3	ZT3	n	$\langle n\rangle_t$:= 3	+		3				B	13	CB	

Bezeichnung	Code	adr	Wirkung	mod2	R	⟨n⟩	$\langle n\rangle_a$	$\langle K\rangle_{s_L}$	$\langle K\rangle_{s_R}$	⟨X⟩	Alarm TK	Werk	Takte	Int.	Bemerkungen
Lösche Speicher	LC	n	⟨n⟩ := +0 $\langle n\rangle_a := \langle n\rangle_a$	+		t_a; +0	$\langle n\rangle_a$					B	13	33	
Lösche markiert	LMT	n	⟨n⟩ := +0 $\langle n\rangle_a$:= L	+		t_a; +0	L				2,3	B	13	32	
Lösche Marke im Speicher	LMC	n	$\langle n\rangle_a$:= 0	+			0				2,3	B	13	31	
Setze Marke im Speicher	ZMC	n	$\langle n\rangle_a$:= L	+			L				2,3	B	13	30	
Lösche und setze Merklichter	LZL	s_L s_R	$\langle K\rangle_{s_R}$:= 0 $\langle K\rangle_{s_L}$:= L					L	0			B	3	10	s: Merklichter 0,1,2,3,4,5,6,7 und 8 0 bedeutet kein Merklicht (0 muß angegeben werden)
Negiere Merklichter	NL	s	$\langle K\rangle_s$:= $\langle K\rangle_s$ invertiert					invertiert				B	3	12	
Setze Indexbasis	ZI	m	⟨X⟩ := $\langle m\rangle_{3-24}$							$\langle m\rangle_{3-24}$		B	17 x + 58	B7	x: Anzahl der Indexregister, die zurückgespeichert werden

Sprünge

Bezeichnung	Code	adr	Wirkung	mod2	R	$\langle F\rangle_{9-24}$	$\langle F\rangle_{1-24}$	$\langle H\rangle$	$\langle B\rangle$	$\langle U\rangle$	U-Alarm	Werk	Takte	Int.	Bemerkungen
Nullbefehl	NULL	z	keine Wirkung		x							B	1	00	Nur übliche Erhöhung des Befehlsfolgeregisters. z: ohne Bedeutung, muß angegeben werden.
Wartebefehl	WB	z	z Uhrimpulse warten (Uhrimpulse alle 10µs									B R	s.Bemerkung.	F8	z: 0...65535 Takte: bei z = 0: 3 Takte bei z ≠ 0: (10z-5)µs
Springe	S	m	$\langle F\rangle_{9-24} := m$	+		m						B	1	36	
Springe nach Ersetzung	SE	m	$\langle F\rangle := \langle m\rangle$ + mod2	sp	x		$\langle m\rangle$ + mod2					B	2	BC	Sprung in and. Großseite mögl.
Springe und bringe $\langle F\rangle$ + 1 nach B	SFB	m	$\langle B\rangle := \langle F\rangle + 1$ $\langle F\rangle_{9-24} := m$	+		m			$\langle F\rangle$ + 1			B	1	3A	
Springe nach Ersetzung und bringe $\langle F\rangle$ + 1 nach B	SFBE	m	$\langle B\rangle := \langle F\rangle + 1$ $\langle F\rangle_{1-24} := \langle m\rangle$ + mod2	sp			$\langle m\rangle$ + mod2		$\langle F\rangle$+1			B R	15	FA	Sprung in andere Großseite mögl.
Springe in Unterprogramm	SU	m	$\langle U\rangle := \langle U\rangle + 1$ $\langle\langle U\rangle\rangle := \langle F\rangle + 1$ $\langle F\rangle_{9-24} := m$	+		m				$\langle U\rangle$ + 1	$\langle U\rangle_{alt} = 254$	B	8	38	$\langle U\rangle$: 0...255 ($\langle U\rangle$: 255 + 1 = 0)
Springe in Unterprogramm nach Ersetzung	SUE	m	$\langle U\rangle := \langle U\rangle + 1$ $\langle\langle U\rangle\rangle := \langle F\rangle + 1$ $\langle F\rangle := \langle m\rangle$ + mod2	sp	x		$\langle m\rangle$ + mod2			$\langle U\rangle$ + 1	$\langle U\rangle_{alt} = 254$	B	8	BD	$\langle U\rangle$: 0...255 ($\langle U\rangle$: 255 + 1 = 0) Sprung in and. Großseite mögl.
Springe ins System und reserviere	SSR	p_L p_R	$\langle F\rangle := \langle a + 6\rangle$ $\langle B\rangle :=$ adr	+		Sprung in den Abwickler zur Erbringung von Dienstleistungen gem. p_L und p_R; s.spezielle Beschreibungen		$\langle a + 6\rangle$	adr			B R	48	BB	p_L, p_R: 0...255 a = $\langle BL\rangle \cdot 2^8$ = Anfangsadresse des Leitblocks

Sprungbedingung	Code	adr		mod2	R	$\langle A\rangle$						Werk	Takte	Int.	Bemerkungen
Springe wenn identisch 0	SIO	m	$\langle A\rangle = 0$	+		m *						B R	1	A4	
Springe wenn nicht 0	SNO	m	$\langle A\rangle \neq 0$	+		m *						B R	1	A7	
Springe wenn größer gleich 0	SGGO	m	$\langle A\rangle \geq 0$	+		m *						B R	1	A6	$\langle A\rangle_t$ = 2 oder 3: Sprung immer!
Springe wenn größer 0	SGO	m	$\langle A\rangle > 0$	+		m *						B R	1	D4	
Springe wenn kleiner gleich 0	SKGO	m	$\langle A\rangle \leq 0$	+		m *						B R	1	A5	
Springe wenn kleiner 0	SKO	m	$\langle A\rangle < 0$	+		m *						B R	1	D5	$\langle A\rangle_t$ = 2 oder 3: Sprung nie!
Springe wenn rechtes Bit in A gesetzt	SR	m	$\langle A\rangle_{48} = L$	+		m *						B R	1	B8	
Springe wenn rechtes Bit in A nicht gesetzt	SRN	m	$\langle A\rangle_{48} = 0$	+		m *						B R	1	BA	
Springe wenn identisch	SI	m	$\langle A\rangle = \langle H\rangle$	+		m *	norm**	norm**				B R	** 6	AC	
Springe wenn nicht identisch	SN	m	$\langle A\rangle \neq \langle H\rangle$	+		m *	norm.**	norm.**				B R	** 6	AD	
Springe wenn größer gleich	SGG	m	$\langle A\rangle \geq \langle H\rangle$	+		m *	norm.**	norm.**				B R	** 6	AF	
Springe wenn größer	SG	m	$\langle A\rangle > \langle H\rangle$	+		m *	norm.**	norm.**				B R	** 6	AB	
Springe wenn kleiner gleich	SKG	m	$\langle A\rangle \leq \langle H\rangle$	+		m *	norm.**	norm.**				B R	** 6	AE	
Springe wenn kleiner	SK	m	$\langle A\rangle < \langle H\rangle$	+		m *	norm.**	norm.**				B R	** 6	AA	

Bezeichnung	Code	adr	Sprungbedingung	mod2	R	⟨F⟩$_{9-24}$	⟨i⟩	⟨M⟩	⟨K⟩$_s$	⟨D⟩	⟨U⟩	Alarm	Werk	Takte *	Takte ***	Int.	Bemerkungen
Springe wenn Index identisch 0	SXI	m	⟨B⟩ = ±0			m *							B	1	1	24	⟨B⟩$_1$ = ⟨B⟩$_v$
Springe wenn Index nicht identisch 0	SXN	m	⟨B⟩ ≠ ±0			m *							B	1	1	27	
Springe wenn Index größer gleich 0	SXGG	m	⟨B⟩ ≥ ±0			m *							B	1	1	25	
Springe wenn Index größer 0	SXG	m	⟨B⟩ > ±0			m *							B	1	1	CE	
Springe wenn Index kleiner gleich 0	SXKG	m	⟨B⟩ ≤ ±0			m *							B	1	1	26	
Springe wenn Index kleiner 0	SXK	m	⟨B⟩ < ±0			m *							B	1	1	CF	
Springe wenn Indexgröße rechtes Bit = L	SXR	m	⟨B⟩$_{24}$ = L			m *							B	1	1	B2	
Springe wenn Indexgröße rechtes Bit nicht L	SXRN	m	⟨B⟩$_{24}$ = 0			m *							B	1	1	B3	
Springe und zähle wenn Index kleiner 0	SZX	p i	⟨i⟩ < ±0			⟨F⟩ + p *	⟨i⟩ + 1 *						B	11	2	0A	⟨i⟩$_1$ = ⟨i⟩$_v$ p: ±0...±127
Springe wenn Marke	SM	m	⟨M⟩ = L	+		m *		0					B	1	1	34	
Springe wenn Marke nicht	SMN	m	⟨M⟩ = 0	+		m *		0					B	1	1	35	
Springe wenn arithmetischer Alarm (BÜ-Alarm)	SAA	m	BÜ-Alarm	+		m *						BÜ-Alarm wird gelöscht	B R	1	1	A9	
Springe wenn Alarm (Typenkennung)	SAT	m	TK-Alarm	+		m *						TK-Alarm wird gelöscht	B R	1	1	A8	
Springe wenn Typenkennung	ST	p s	⟨s$_2$⟩$_t$ = s$_1$	+		⟨F⟩ + p *							B R	10	3	90	p: ±0...±127 s$_1$: 0,1,2 oder 3 (TK) s$_2$: A,Q,D oder H
Springe wenn Typenkennung nicht	STN	p s	⟨s$_2$⟩$_t$ ≠ s$_1$	+		⟨F⟩ + p *							B R	10	3	91	
Springe wenn Bit gesetzt	SBIT	p s	⟨s$_2$⟩$_{s_1}$ = L			⟨F⟩ + p *	Tabelle für Zeitberechnung: s$_1$ | 1 2 ... 14 15 16 17 ... 31 32 33 ... 47 48 q | 16 15 ... 3 2 1 16 ... 2 1 16 ... 2 1						B R	7+2q		F9	s$_1$: Bitnummer 1...48 s$_2$: Register A, Q, D oder H p : Sprungweite ±0...±127 q : s. Tab. für Zeitberechn.
Springe wenn Exponent größer gleich	SEGG	p$_L$ p$_R$	⟨A⟩$_{41-48}$ ≥ p$_R$	+		⟨F⟩ + p$_L$ *				t$_A$;⟨A⟩		falls ⟨A⟩$_t$≠0: TK-Alarm	B R	10	4	93	p$_L$: Sprungweite p:±0...±127 p$_R$: Vergl.-Exponent ****
Springe wenn Merklicht	SL	p s	eines der ⟨K⟩$_s$ = L			⟨F⟩ + p *							B	9	1	1E	
Springe wenn Merklicht und lösche	SLL	p s	eines der ⟨K⟩$_s$ = L			⟨F⟩ + p *			0				B	9	1	1F	p: ±0...±127
Springe wenn Merklicht nicht	SLN	p s	alle ⟨K⟩$_s$ = 0			⟨F⟩ + p *							B	9	1	1C	s: Merklichter 1,2,... und 8
Springe wenn Merklicht nicht sonst lösche	SNL	p s	alle ⟨K⟩$_s$ = 0			⟨F⟩ + p *			0				B	9	1	1D	

Bezeichnung	Code	adr	Sprungbedingung	mod2	R	⟨F⟩$_{9-24}$	⟨A⟩	⟨Y⟩	Werk	Takte *	Takte ***	Int.	Bemerkungen
Prüfe Dreierprobe und springe wenn richtig	PDP	n	DP = richtig	+		⟨F⟩ + 2 *	t$_n$;⟨n⟩	⟨n⟩$_{5-6}$OL,DP,⟨n⟩$_t$	R B	21	21	B9	DP: Dreierprobenbits Transp. unverän. bei jeder TK DP nicht richt.: kein DP-Alarm

* bei erfüllter Sprungbedingung

** wenn ⟨A⟩$_t$ = ⟨H⟩$_t$ = 0: ⟨A⟩ := ⟨A⟩ normalisiert
⟨H⟩ := ⟨H⟩ normalisiert
ist der Exponent +0 wird er zu -0, zur Ausführungszeit kommt ein Takt hinzu

*** bei nicht erfüllter Sprungbedingung

**** 0...127 (positiv), NO...N127 (negativ)

Modifizieren

Bezeichnung	Code	adr	Wirkung	mod2	R	mod1	$mod2_{neu}$	⟨B⟩	⟨i⟩	Werk	Takte	Int.	Bemerkungen
Modifiziere in jedem Fall	MF	i	mod1 := ⟨i⟩ + mod2 ⟨B⟩ := ⟨i⟩ + mod2	sp		⟨i⟩ *	0	⟨i⟩ *		B	1	0B	
Modifiziere in jedem Fall mit unveränd. B	MFU	i	mod1 := ⟨i⟩ + mod2	sp		⟨i⟩ *	0			B	1	08	Bits 9 - 11 = 00L
Modifiziere aus Speicher in jedem Fall	MCF	m	mod1 := ⟨m⟩ + mod2 ⟨B⟩ := ⟨m⟩ + mod2	sp	x	⟨m⟩ *	0	⟨m⟩ *		B	2	16	
Modifiziere aus Speicher in jedem Fall mit unverändertem B	MCFU	m	mod1 := ⟨m⟩ + mod2	sp	x	⟨m⟩ *	0			B	1	3D	
Modifiziere doppelt	MD	i_L i_R	mod1 := ⟨i_R⟩ + mod2 mod2 := ⟨i_L⟩ ⟨B⟩ := ⟨i_L⟩	sp		⟨i_R⟩ *	⟨i_L⟩	⟨i_L⟩		B	5	09	
Modifiziere	M	i	⟨B⟩ := ⟨i⟩ + mod2 mod2 := ⟨i⟩ + mod2	sp			⟨i⟩ *	⟨i⟩ *		B	1	08	Bits 9 - 11 = L00
Modifiziere nach Erhöhung	MH	p i	mod2 := ⟨i⟩ + p ⟨B⟩ := ⟨i⟩ + p ⟨i⟩ := ⟨i⟩ + p				⟨i⟩ + p	⟨i⟩ + p	⟨i⟩ + p	B	10,5	2D	p: ±0...±127
Modifiziere nach Erhöhung um Indexzelle	MHX	i_L i_i	mod2 := ⟨i_L⟩ + ⟨i_R⟩ ⟨B⟩ := ⟨i_L⟩ + ⟨i_R⟩ ⟨i_R⟩ := ⟨i_L⟩ + ⟨i_R⟩				⟨i_L⟩ + ⟨i_R⟩	⟨i_L⟩ + ⟨i_R⟩	⟨i_R⟩ := ⟨i_L⟩ + ⟨i_R⟩	B	14,5	0E	
Modifiziere mit Register und Indexzelle	MRX	s i	mod2 := ⟨i⟩ ± ⟨s_1⟩ ⟨B⟩ := ⟨i⟩ ± ⟨s_1⟩ falls s_3 = C: ⟨i⟩ := ⟨i⟩ ± ⟨s_1⟩				⟨i⟩ ± ⟨s_1⟩	⟨i⟩ ± ⟨s_1⟩	falls s_3 = C: ⟨i⟩ ± ⟨s_1⟩	B	12	8D **	s_1: A,Q,D,H (rechtes Halbwort) oder B s_2: leer: + / N : - } statt ± s_3: leer: nicht speichern / C : zurückspeichern
Modifiziere aus Speicher	MC	m	⟨B⟩ := ⟨m⟩ + mod2 mod2 := ⟨m⟩ + mod2	sp	x		⟨m⟩ *	⟨m⟩ *		B	2	14	
Modifiziere aus Speicher nach Ersetzungen	MCE	m	⟨B⟩ :=⟨(...⟨m⟩...)⟩+mod2 mod2:=⟨(...⟨m⟩...)⟩+mod2	sp	x		⟨(...⟨m⟩...)⟩* das 1. Bit wird dem 2. angeglichen	⟨(...⟨m⟩...)⟩*		B	13x -9	17	x: Anzahl der Halbwörter in der Ersetzkette Abbruch wenn: ⟨(...⟨m⟩...)⟩$_1$ = L
Modifiziere mit Adressenteil	MA	z	⟨B⟩ := z + mod2 mod2 := z + mod2	sp			z *	z *		B	1	03	z: 0...65 535
Modifiziere mit negativem Adressenteil	MNA	z	⟨B⟩ :=-z + mod2 mod2 :=-z + mod2	sp			-z *	-z *		B	2	02	

* Ist vom vorhergehenden Befehl ein Modifikator 2. Art vorhanden, so wird er addiert

** siehe Internspezifikation auf Seite 23

Ersetzen (und modifizieren)

Bezeichnung	Code	adr	Wirkung	mod2	R	op	adr	⟨B⟩	⟨i⟩	$mod2_{neu}$	Werk	Takte	Int.	Bemerkungen
Ersetze	E	c i	adr := ⟨i⟩ mod2 := mod2	sp		c	⟨i⟩			mod2	B	4	29	
Ersetze zählend	EZ	c i	adr := ⟨i⟩ + 2 + mod2 ⟨B⟩ := ⟨i⟩ + 2 ⟨i⟩ := ⟨i⟩ + 2	sp		c	⟨i⟩ + 2 *	⟨i⟩ + 2	⟨i⟩ + 2	0	B	5	2B	
Ersetze negativ zählend	ENZ	c i	adr := ⟨i⟩ + mod2 ⟨B⟩ := ⟨i⟩ - 2 ⟨i⟩ := ⟨i⟩ - 2	sp		c	⟨i⟩ *	⟨i⟩ - 2	⟨i⟩ - 2	0	B	8	2A	
Ersetze und modifiziere mit B	EMB	c i	adr := ⟨i⟩ mod2 := ⟨B⟩			c	⟨i⟩			⟨B⟩	B	4	28	
Modifiziere Adressenteil mit B	MAB	c p	adr := ⟨B⟩ + p ⟨B⟩ := ⟨B⟩ + p			c	⟨B⟩ + p	⟨B⟩ + p		0	B	9,5	20	p: ±0...±127
Modifiziere Adressenteil mit B bei Invarianz der Sprungadresse	MABI	c p	adr := ⟨B⟩ + p ⟨B⟩ := ⟨B⟩ + p			c	⟨B⟩ + p	⟨B⟩ + p		0	B	9,5	3F	p: ±0...±127 ****
Modifiziere über U	MU	c p	adr := ⟨⟨U⟩⟩ + p			c	⟨⟨U⟩⟩ + p			0	B	14,5	05	⟨U⟩ := ⟨U⟩-1 wenn c Sprungbef. u. Bedingung erfüllt ist** p: ±0...±127
Ersetze nach Modifizierung über U	EMU	c p	adr := ⟨⟨⟨U⟩⟩ + p⟩ mod2 := mod2	sp		c	⟨⟨⟨U⟩⟩ + p⟩			mod2	B	29,5	04	p: ±0...±127
Relativ-Adressierung mit Registerinhalt	RLR	c s	adr := ⟨s⟩ + ⟨F⟩ mod2 := mod2	sp		c	⟨s⟩ + ⟨F⟩	falls s = F: adr := 2·⟨F⟩ + 1		mod2	B R	10	E0	s: A, Q, D, H = rechte Hälfte AL,QL,DL,HL = linke Hälfte F,B, U,Y
Registeradressierung	R	c s	operand := ⟨s_1⟩	+		c	⟨s_1⟩ = Reg.-Operand:	s_1 = A,Q,D oder H: ro := t_{s_1};⟨s_1⟩***			B R	4 / 6	96	s_2: L : linke 24 Bits : leer : rechte 24 Bits c : erlaubter Code ro: Registeroperand
								s_1 = Y,U : ro := 1;0,⟨s_1⟩						
								s_1 = B : ro := 1;v,⟨B⟩						
								s_1 = F : ro := 1;v,⟨F⟩+1						
Tue	T	m	op, adr := ⟨m⟩ mod2 := mod2	sp	x	⟨m⟩$_{1-8}$	0,⟨m⟩$_{9-24}$			mod2	B	5	CC	⟨m⟩$_t$: beliebig

* Ist vom vorhergehenden Befehl ein Modifikator 2. Art vorhanden, so wird er addiert

** Sprung in eine andere Großseite ist möglich

***bei s_1 = A,Q,D oder H und ⟨s_1⟩$_t$ = 0 oder 1:
⟨s_1⟩$_1$:= ⟨s_1⟩$_a$
⟨s_1⟩$_2$:= ⟨s_1⟩$_v$

**** für c die Sprungbefehle von Seite 12 u. 13 (mit Ausnahme der Befehle NULL, PDP, SSR und WB)
bei erfüllter Sprungbedingung ist ein Sprung in eine andere Großseite möglich.

Als Zweitcodes (c) sind alle Befehlscodes zugelassen.
Bei dem Befehl R jedoch nur:

A A2 AB AQ AT AU AUT

B B2 B2V B2VN B3 B3V BB BD BH BN BNR BQ BQB BR BT BU

DV DVD DVI

ET

GA GAB GDV GDVI GMAN GML GMLA GMLN GSB GSBB GSBD GSBI

HBC

M2 M2N M2NR M2R MAN MANR MAR MC MCE MCF MCFU ML MLA MLD MLN MLR MNR

NULL

REZ

SB SB2 SBB SBD SBI SBQ SBT SBU SE SUE

T TCB

VBC VEL

ZUS

Aufbereitung

Bezeichnung	Code	adr	Wirkung	mod2	R	Voraussetzung	⟨A⟩	⟨B⟩	⟨Q⟩	⟨Y⟩	Werk	Takte	Int.	Bemerkungen
Umschlüsseln	US	s i	⟨A⟩ := ⟨A⟩umgeschlüsselt	sp		⟨A⟩ = umzuschl. Zeich. ⟨i⟩+mod2 = Anfangsadr. der Umschl.-Tabelle *	t_n;⟨A⟩umgeschl. falls s_2=E $\langle A\rangle_{1-x}$:= 0	⟨i⟩+mod2 Adresse der Umschl.-Tabelle	falls s_2 = G t_n;⟨A⟩ umgeschl.	falls s_2 = G Anzahl der umgeschlüsselt. Zeich.	B R	s_2=E:32 s_2=G:und s_1=6:176 =8:136 =C: 96	E3	s_1: Bitlänge der Zeichen 6, 8 oder C (12 Bits) s_2: E = ein Zeichen rechtsbünd. G = alle Zeich. 8, 6 oder 4 t_n: Typenkennung des Wortes, in dem das linke Zeichen steht
Invertiere Register	IR	s	⟨s⟩ := ¬⟨s⟩ ⟨s⟩ := |⟨s⟩|			falls s_2: leer, invertiere falls s_2: B, bilde Betrag im Register					R	4p + 1	E1	s_1: Register A,Q,D und H p : Anzahl der adr. Register
Schifte	SH	s p	Schift gemäß Spezifikation um p Stellen	+		Spezifikation: s_1: A = Register A Q = Register Q AQ = Register A und Q getrennt Z = doppelt langes Register A,Q s_2: leer, L = Rechtsschift, Linksschift s_3: leer, K = gestreckter Schift, Kreisschift s_4: leer, R = ohne Rundung, mit Rundung s_5: leer, U = abhängig von TK, unabh. von TK s_6: leer, B = nicht zählen, zählen der aus A geschifteten 'L'-Bits im Reg.Y				BÜ Alarm ≷ 1)	R	2(q+r)+5 p:4=q Rest r zusätzl. 5 wenn s_4=R u.gerundet wurde	9B	1) BÜ Alarm möglich: nur bei Linksschift in A mit TK=0. oder 1; bei Rechtsschift höchstens um 1 Stelle mit Rundung p= Anzahl der Schiftschritte ±0...+127 p:4 = q Rest r
Schifte in B	SHB	s p	⟨B⟩ := ⟨B⟩ geschiftet um p Stellen nach rechts oder links mit Nachziehen von 0-Bits			s: R = Schift nach rechts L = Schift nach links					B	4*p+6	21	p: Anzahl der Schiftschritte 0...255
Vorzeichenangleich zwischen A und Q	VAQ	z	⟨A,Q⟩ := ⟨A⟩ + ⟨Q⟩ • 2^{-48} $\langle Q\rangle_v$:= $\langle A\rangle_1$ falls ⟨A⟩ = ±0, ⟨Q⟩ ≠ ±0 $\langle A\rangle_v$:= $\langle Q\rangle_1$	+		VAQ gleicht die Vorzeichen an und berichtigt das Ergebnis unter Erhaltung des Zahlenwertes				Alarm Y / BÜ / TK: $\langle A\rangle_t \neq 1$	R	2 bei $\langle A\rangle_1 = \langle Q\rangle_v$ 17 bei $\langle A\rangle_1 \neq \langle Q\rangle_v$	63	Adressenteil z ist ohne Bedeutung, muß aber angegeben werden
Normalisiere	NRM	s	bei s = G: ⟨A⟩ := ⟨A⟩normalisiert	+			4t			≠0	R	2t + 3	9F	s = G, FG, F, F4, N oder L t: Anzahl der Tetradenschifte e: Anzahl der Einerschifte
			bei s = FG: $\langle A\rangle_{Gleitk.}$:=⟨A,Q⟩$_{Festk.}$			Die Festkommazahl wird als echter Bruch aufgefaßt $\langle A\rangle_t$:= 0;⟨A,Q⟩gerundet, ⟨Q⟩ := 1;+0	+0	$\langle A\rangle_1 \neq \langle Q\rangle_1$		≠1		2t + 10		
			bei s = F: ⟨A,Q⟩ := ⟨A,Q⟩normalisiert				4t	$\langle A\rangle_1 \neq \langle Q\rangle_1$		≠1		2t + 7		
			bei s = F4: ⟨A,Q⟩ := ⟨A,Q⟩normalisiert				t	$\langle A\rangle_1 \neq \langle Q\rangle_1$		≠1		2t + 7		
			bei s = N falls ⟨A⟩ ≠ +0 ⟨A⟩ := ⟨A⟩ links geschift. bis $\langle A\rangle_1$ = L			beim Schiften werden Nullen nachgezogen falls ⟨A⟩ = +0: ⟨A⟩ := ⟨A⟩	e 48					2e + 3		
			bei s = L: ⟨A⟩ := ⟨A⟩ links geschift. bis $\langle A\rangle_1$ = 0			beim Schiften werden Nullen nachgezogen	e							

*Tabelle in Viertelwörtern.
Die Anfangsadresse ist eine Halbwortadresse

Bezeichnung	Code	adr	Wirkung	mod2	R	Voraussetzung		Alarm Y	Alarm BÜ	Alarm TK	Werk	Takte	Int.	Bemerkungen
Konvertiere Dezimalzahl in Festkommazahl rechtsbündig	KDFR	p	$\langle A\rangle_{Festk.} := \langle A,Q\rangle_{Dez.}$	+		$\langle A,Q\rangle$ = pos. Dez.-Zahl Komma rechts	$\langle A\rangle := 1;\langle A,Q\rangle\ \langle D\rangle := 1;+0$ $\langle Q\rangle := 1;+0\quad \langle H\rangle := 1;+0$	+0			R	7p + 22	94	p: 1...13 Anzahl der Dezimalstellen, die konvertiert werden
Konvertiere Festkommazahl linksbündig in Dezimalzahl	KFLD	p	$\langle A,Q\rangle_{Dez.} := \langle A\rangle_{Festk.}$	+		$\langle A\rangle$ = pos. Festk.-Zahl Komma links $\langle A\rangle_t = 0$ oder 1 $\langle A\rangle_1 = \langle A\rangle_2 = 0$	Im doppelt langen Register $\langle A,Q\rangle$ steht rechtsbündig die Dezimalzahl jeweils mit Typenkennung = 1. Im Reg. D steht mit TK = 1 der Rest	+0	≷	2,3	R	$\langle A\rangle > 0$ 9p + 5	95	p: 1...13 Anzahl der gewünschten Dezimalstellen

* Tabelle in Viertelwörtern
Die Anfangsadresse ist eine Halbwortadresse

Tabelle durchsuchen

	Code	adr	Suchbedingung			Eingangsgrößen	⟨B⟩	⟨A⟩	⟨Q⟩	⟨D⟩	Alarm BÜ	Alarm TK					
Tabelle durchsuchen auf Identität	TLI	n	$\langle n+2k\rangle = \langle D\rangle$ ** Tabellenende: TK ≠ t_D			$\langle D\rangle$ =Suchwort $\langle D\rangle_1 = \langle D\rangle_2$ bei $\langle B\rangle_t$=0 oder 1	n*	$t_D;\langle D\rangle$***		***			≠ $\langle D\rangle_t$	B R	15p + 10	EC	k: 0,1,2,... p: Anzahl der untersuchten Worte in der Tabelle x: 1...48 falls Dehnungswert $\langle B\rangle = \pm 0$ Wirkung wie Nullbefehl
Tabelle durchsuchen mit Dehnung	TLD	n	$\langle n+k\langle B\rangle\rangle \geq \langle D\rangle$ ** Tabellenende: TK ≠ t_D			$\langle D\rangle$ =Suchwort $\langle D\rangle_1 = \langle D\rangle_2$ bei $\langle D\rangle_t$=0 oder 1 $\langle B\rangle$=Dehnungswert	n*	$t_D;\langle D\rangle$***		***			≠ $\langle D\rangle_t$	B R	14,5p + 10,5	EB	
Tabelle durchsuchen mit Dehnung und Maske	TDM	n	$\langle n+k\langle B\rangle\rangle_x = \langle D\rangle_x$ für $\langle H\rangle_x = 0$ ** Tabellenende: TK ≠ t_D			$\langle D\rangle$ =Suchwort $\langle H\rangle$ =Maske $\langle B\rangle$=Dehnungswert	n*	$t_H;\langle H\rangle\wedge\langle n^*\rangle$					≠ $\langle D\rangle_t$	B R	14,5p + 2,5	EA	
Tabelle durchsuchen auf Maximum	TMAX	n	$\langle n+k\langle B\rangle\rangle_x$ = Max. für $\langle H\rangle_x = 0$ Tabellenende: TK ≠ t_n			$\langle H\rangle$=Maske $\langle B\rangle$=Dehnungswert	n*	$t_n{}^*;\langle H\rangle\wedge\langle n^*\rangle$		$t_n{}^*;\neg\langle H\rangle\wedge\langle n^*\rangle$				B R	17p + 4	EF	falls Dehnungswert $\langle B\rangle = \pm 0$ Wirkung wie Nullbefehl
Tabelle durchsuchen auf Minimum	TMIN	n	$\langle n+k\langle B\rangle\rangle_x$ = Min. für $\langle H\rangle_x = 0$ Tabellenende: TK ≠ t_n			$\langle H\rangle$=Maske $\langle B\rangle$=Dehnungswert	n*	$t_n{}^*;\langle H\rangle\wedge\langle n^*\rangle$		$t_n{}^*;\neg\langle H\rangle\wedge\langle n^*\rangle$				B R	17p + 4	EE	
Tabelle durchsuchen logarithmisch	TLOG	n	$\langle n+2k\rangle_x \geq \langle D\rangle_x$ Voraussetzung: $\langle n\rangle_x < \langle D\rangle_x$ } für $\langle H\rangle_x = 0$			$\langle A\rangle$=Tabellenlänge in Ganzwört. $2 \leq \langle A\rangle \leq 2^{20}$ $\langle D\rangle$=Suchwort $\langle H\rangle$=Maske	n^+	$t_n{}^+;\langle H\rangle\wedge\langle n^+\rangle$ falls kein Wort gefund. 3;0		$t_D;\neg\langle H\rangle\wedge\langle D\rangle$		kein Wort gef.		B R	23q + 46	ED	$\langle F\rangle := \langle F\rangle_{9-24} + 2$ falls $\langle n+2k\rangle_x > \langle D\rangle_x$ oder kein Wort gefunden q : $({}_2\log\langle A\rangle)-1 \leq q < {}_2\log\langle A\rangle$ q : ganzzahlig

* Adresse des zuerst gefundenen Wortes, das den Suchbedingungen genügt oder Adresse des ersten Wortes hinter der Tabelle (bei TMAX und TMIN immer Adresse des zuerst gefundenen Wortes)

** Wird kein Wort gefunden: Abbruch mit TK-Alarm

*** falls $\langle D\rangle_t = 0$: $\langle D\rangle := \langle D\rangle$ normalisiert, Vergleichsoperand normalisiert (im Speicher unverändert)

n^+ niedrigste Adresse der nach den Suchbedingungen gefundenen Wörter oder erstes Wort hinter der Tabelle

Zentralcode

Stand 1972

0000	000L	00L0	00LL	0L00	0L0L	0LL0	0LLL	L000	L00L	L0L0	L0LL	LL00	LL0L	LLL0	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0 NUL	16	32 SUB	48 BEL	64	80	96 "	112 %	128 ^	144 +	160 (	176 0	192 A	208 Q	224 a	240 q	0	0000
1	17	33 EM	49 DC1	65	81	97 '	113 §	129 v	145 –	161)	177 1	193 B	209 R	225 b	241 r	1	000L
2	18	34 CAN	50 DC2	66	82	98 ´	114 #	130 ¬	146	162 [	178 2	194 C	210 S	226 c	242 s	2	00L0
3	19	35	51 DC3	67	83	99 `	115 $(¤)	131	147 /	163]	179 3	195 D	211 T	227 d	243 t	3	00LL
4	20	36	52 DC4	68	84	100	116 ¢	132	148	164	180 4	196 E	212 U	228 e	244 u	4	0L00
5	21 NL	37	53 FL	69	85	101 ^	117	133	149	165	181 5	197 F	213 V	229 f	245 v	5	0L0L
6 SOH	22 CR	38	54	70	86	102 o	118 @	134 †	150	166 ⟨	182 6	198 G	214 W	230 g	246 w	6	0LL0
7 STX	23 NF	39	55	71	87	103 ~	119 &	135	151 =	167 ⟩	183 7	199 H	215 X	231 h	247 x	7	0LLL
8 ETX	24 VT	40	56	72	88	104	120 *	136	152 ≠	168 (MZ) }	184 8	200 I	216 Y	232 i	248 y	8	L000
9 EOT	25	41	57	73	89	105	121	137	153	169 .	185 9	201 J	217 Z	233 j	249 z	9	L00L
10 ENQ	26	42	58	74	90	106	122	138	154	170 ,	186	202 K	218 Ä	234 k	250 ä	A	L0L0
11 ACK	27	43 HT	59	75	91	107 \	123	139 |	155 <	171 :	187	203 L	219 Ö	235 l	251 ö	B	L0LL
12 DLE	28	44 BS	60 IS4	76	92	108 (	124 ¤	140	156 >	172 ;	188	204 M	220 Ü	236 m	252 ü	C	LL00
13 NAK	29	45 ESC	61 IS3	77	93	109)	125	141 10	157 ≦	173 !	189	205 N	221	237 n	253 ß	D	LL0L
14 SYN	30	46 SO	62 IS2	78	94	110 _	126	142	158 ≧	174 ?	190	206 O	222	238 o	254	E	LLL0
15 ETB	31	47 SI	63 IS1	79	95	111 —	127 π	143	159	175 SP	191 (PZ) {	207 P	223	239 p	255 DEL	F	LLLL

Potenzen von 2

2^n	n	2^{-n}
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0. 244 140 625
8 192	13	0. 122 070 312 5
16 384	14	0. 061 035 156 25
32 768	15	0. 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0. 007 629 394 531 25
262 144	18	0. 003 814 697 265 625
524 288	19	0. 001 907 348 632 812 5
1 048 576	20	0. 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0. 238 418 579 101 562 5
8 388 608	23	0. 119 209 289 550 781 25
16 777 216	24	0. 059 604 644 775 390 625
33 554 432	25	0. 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0. 007 450 580 596 923 828 125
268 435 456	28	0. 003 725 290 298 461 914 062 5
536 870 912	29	0. 001 862 645 149 230 957 031 25
1 073 741 824	30	0. 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0. 232 830 643 653 869 628 906 25
8 589 934 592	33	0. 116 415 321 826 934 814 453 125
17 179 869 184	34	0. 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0. 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0. 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0. 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0. 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0. 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0. 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0. 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0. 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0. 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0. 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0. 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0. 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0. 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.(15) 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0. 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0. 111 022 302 462 515 654 042 363 166 809 062 031 25
18 014 398 509 481 984	54	0. 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0. 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.(15) 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0. 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0. 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0. 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0. 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625
2 305 843 009 213 693 952	61	0.(18) 433 680 868 994 201 773 602 981 120 347 976 684 570 312 5
4 611 686 018 427 387 904	62	0. 216 840 434 497 100 886 801 490 560 173 988 342 285 156 25
9 223 372 036 854 775 808	63	0. 106 420 217 248 550 443 400 745 280 086 994 171 142 578 125
18 446 744 073 709 551 616	64	0. 054 210 108 624 275 221 700 372 640 043 497 085 571 289 062 5
36 893 488 147 419 103 232	65	0. 027 105 054 312 137 810 850 186 320 021 748 542 785 644 531 25
73 786 976 294 838 206 464	88	0.(18) 013 552 527 156 068 805 425 093 160 010 874 271 392 822 265 625
147 573 952 589 676 412 928	67	0. 006 776 263 578 034 402 712 546 580 005 437 135 696 411 132 812 5
295 147 905 179 352 825 856	68	0. 003 388 131 789 017 201 356 273 290 002 718 567 848 205 566 406 25
590 295 810 358 705 651 712	69	0. 001 694 065 894 508 600 678 136 645 001 359 283 924 102 783 203 125
1 180 591 620 717 411 303 424	70	0. 000 847 032 947 254 300 339 068 322 500 679 641 962 051 391 601 582 5
2 361 183 241 434 822 606 848	71	0.(21) 423 516 473 627 150 169 534 161 250 339 820 981 025 695 800 781 25
4 722 366 482 869 645 213 696	72	0. 211 758 236 813 575 084 767 080 625 169 910 490 512 847 900 390 625
9 444 732 965 739 290 427 392	73	0. 105 879 118 406 787 542 383 540 12 584 955 245 256 423 950 195 312 5
18 889 465 931 478 580 854 784	74	0. 052 939 559 203 393 771 191 770 156 292 477 622 628 211 975 097 656 25
37 778 931 862 957 161 709 568	75	0. 026 469 779 601 696 885 595 885 078 146 238 811 314 105 987 548 828 125
75 557 863 725 914 323 419 136	76	0.(21) 013 234 889 800 848 442 797 942 539 073 119 405 657 052 993 774 414 062 5
151 115 727 451 828 646 838 272	77	0. 006 617 444 900 424 221 398 971 269 536 559 702 828 526 496 887 207 031 25
302 231 454 903 657 293 676 544	78	0. 003 308 722 450 212 110 699 485 634 768 279 851 414 263 248 443 603 515 625
604 462 909 807 314 587 353 088	79	0. 001 654 361 225 106 055 349 742 817 384 139 925 707 131 624 221 801 757 812 5
1 208 925 819 614 629 174 706 176	80	0 000 827 180 612 553 027 674 871 408 692 069 962 853 565 812 110 900 878 906 25
2 417 851 639 229 258 349 412 352	81	0.(24) 413 590 306 276 513 837 435 704 346 034 981 426 782 906 055 450 439 453 125
4 835 703 278 458 516 698 824 704	82	0. 206 795 153 138 256 918 717 852 173 017 490 713 391 453 027 725 219 726 562 5
9 671 406 556 917 033 397 649 408	83	0. 103 397 576 569 128 459 358 926 086 508 745 356 695 726 513 862 609 863 281 25
19 342 813 113 834 066 795 298 816	84	0. 051 698 788 284 564 229 679 463 043 254 372 678 347 863 256 931 304 931 640 625
38 685 626 227 668 133 590 597 632	85	0. 025 849 394 142 282 114 839 731 521 627 186 339 173 931 628 465 652 465 820 312 5
77 371 252 455 336 267 181 195 264	86	0.(24) 012 924 697 071 141 057 419 865 760 813 593 169 586 965 814 232 826 232 910 156 25
154 742 504 910 672 534 362 390 528	87	0. 006 462 348 535 570 528 709 932 880 406 796 584 793 482 907 116 413 116 455 078 125
309 485 009 821 345 068 724 781 056	88	0. 003 231 174 267 785 264 354 966 440 203 398 292 396 741 453 558 206 558 227 539 062 5
618 970 019 642 690 137 449 562 112	89	0. 001 615 587 133 892 632 177 483 220 101 699 146 198 370 726 779 103 279 113 769 531 25
1 237 940 039 285 380 274 899 124 224	90	0. 000 807 793 566 946 316 088 741 610 050 849 573 099 185 363 389 551 639 556 884 765 625
2 475 880 078 570 760 549 798 248 448	91	0.(27) 403 896 783 473 158 044 370 805 025 424 786 549 592 681 694 775 819 778 442 382 812 5
4 951 760 157 141 521 099 596 496 896	92	0. 201 948 391 736 579 022 185 402 512 712 393 274 796 340 847 387 909 889 221 191 406 25
9 903 520 314 283 042 199 192 993 792	93	0. 100 974 195 868 289 511 092 701 256 356 196 637 398 170 423 693 954 944 610 595 703 125
19 807 040 628 566 084 398 385 987 584	94	0. 050 487 097 934 144 755 546 350 628 178 098 318 699 085 211 846 977 472 305 297 851 562 5
39 614 081 257 132 168 796 771 975 168	95	0. 025 243 548 967 072 377 773 175 314 089 049 159 349 542 605 923 488 736 152 648 925 781 25
79 228 162 514 264 337 593 543 950 336	96	0 (27) 012 621 774 483 536 188 886 587 657 044 524 579 674 771 302 961 744 368 076 324 462 890 625
158 456 325 028 528 675 187 087 900 672	97	0. 006 310 887 241 768 094 443 293 828 522 262 289 837 385 651 480 872 184 038 162 231 445 312 5
316 912 650 057 057 350 374 175 801 344	98	0. 003 155 443 620 884 047 221 646 914 261 131 144 918 692 825 740 436 092 019 081 115 722 658 25
633 825 300 114 114 700 748 351 602 688	99	0. 001 577 721 810 442 023 610 823 457 130 565 572 459 346 412 870 218 046 009 540 557 861 328 125
1 267 650 600 228 229 401 496 703 205 376	100	0. 000 788 860 905 221 011 805 411 728 565 282 786 229 673 206 435 109 023 004 770 278 930 664 062 5

(.) = A(der Nullen

Befehlscode Intern - Extern

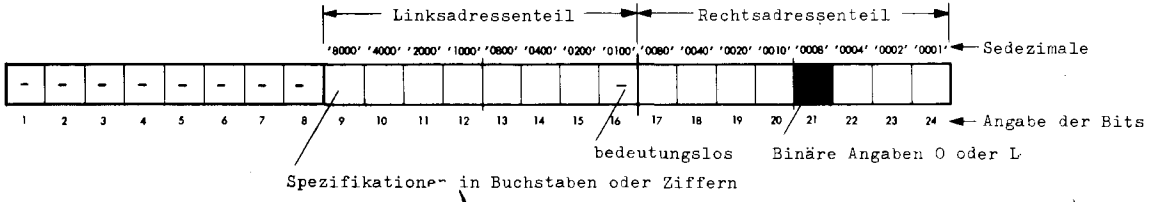
0000	000L	00L0	00LL	0L00	0L0L	0LL0	0LLL	L000	L00L	L0L0	L0LL	LL00	LL0L	LLL0	LLLL
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
NULL	LZL	MAB	ZMC	AB		DV	B	C	ST	C2	BL		CU	RLR	DA
XBA	HBA	SHB	LMC	SBB		DVD	BD	CR	STN	C3	VPU *		BZN	IR	DSB
MNA	NL	WTV	LMT	A	GAB	DVI	BQ	CMT	ZTR	CMC	SXR		ZDP *		DML
MA	VBA	WTR	LC	AC	GSBB	VAQ	BH	CMR	SEGG	BC	SXRN		BU	US	MLD
EMU	MC	SXI	SM	SBI	ML	GDV	BB	CN	KDFR	SI0	Y *		SG0		AT
MU	VBC	SXGG	SMN	SBD	MLR	REZ	BN	CB	KFLD	SKG0	LEI *		SK0		SBT
BCL	MCF	SXKG	S	SB	MLA	GDVI	BR	CD	R	SGG0	BCI			BNZ	BT
TBC	MCE	SXN	VSS *	SBC	MAR	B2VN	BNR	CQ	RT	SN0	ZI			CNZ	CT
MFU** M, XB	XC ** XCN	EMB	SU	GSBI	MLN	VEL	M2N	VLA	AA	SAT	SR	ZT0	BZ2		WB
MD	XBAN	E	TCB	AU	MNR	AUT	M2NR	ATA	SBA	SAA	PDP	ZT1	BZ		SBIT
SZX	ZX	ENZ	SFB	GAC	MAN	ET	M2	ETA	LR	SK	SRN	ZT2	BQB	TDM	SFBE
MF	SW *	EZ		GA	MANR	ZUS	M2R	LA	SH	SG	SSR	ZT3	CZ	TLD	BSS
TXX	SLN	HXP	HBC	GSBD	GMLN	B3	A2	TXR	TRX	SI	SE	T	BAR	TLI	ZK
TTX	SNL	MH	MCFU	SBU	GMAN	B3V	SB2	RX ** MRX	HALT *	SN	SUE		BANR	TLOG	TOK
MHX	SL	HXX	ZU	GSBC	GML	B2	AQ	BA		SKG	BLEI	SXG		TMIN	QBR
HBPX	SLL	VXX	MABI	GSB	GMLA	B2V	SBQ	CH	NRM	SGG	VMO *	SXK	BAN	TMAX	QCR

GR 1

2^7 1. Sedezimale	2^0 2. Sedezimale
0	0000
1	000L
2	00L0
3	00LL
4	0L00
5	0L0L
6	0LL0
7	0LLL
8	L000
9	L00L
A	L0L0
B	L0LL
C	LL00
D	LL0L
E	LLL0
F	LLLL

*nicht für die Programmierung von Operatoren. Der Befehl SSR findet in Abwicklerdiensten Verwendung. **Unterscheidung im Adressenteil (siehe Internspezifikationen)

Spezifikationen - Intern



Konvertierungstafel

sedezimal	dezimal
100 000	1 048 576
200 000	2 097 152
300 000	3 145 728
400 000	4 194 304

010 000	65 536
020 000	131 072
030 000	196 608
040 000	262 144
050 000	327 680
060 000	393 216
070 000	458 752
080 000	524 288
090 000	589 824
0A0 000	655 360
0B0 000	720 896
0C0 000	786 432
0D0 000	851 968
0E0 000	917 504
0F0 000	983 040

001 000	4 096
002 000	8 192
003 000	12 288
004 000	16 384
005 000	20 480
006 000	24 576
007 000	28 672
008 000	32 768
009 000	36 864
00A 000	40 960
00B 000	45 056
00C 000	49 152
00D 000	53 248
00E 000	57 344
00F 000	61 440

000 100	256
000 200	512
000 300	768
000 400	1 024
000 500	1 280
000 600	1 536
000 700	1 792
000 800	2 048
000 900	2 304
000 A00	2 560
000 B00	2 816
000 C00	3 072
000 D00	3 328
000 E00	3 584
000 F00	3 840

000 010	16
000 020	32
000 030	48
000 040	64
000 050	80
000 060	96
000 070	112
000 080	128
000 090	144
000 0A0	160
000 0B0	176
000 0C0	192
000 0D0	208
000 0E0	224
000 0F0	240

000 001	1
000 002	2
000 003	3
000 004	4
000 005	5
000 006	6
000 007	7
000 008	8
000 009	9
000 00A	10
000 00B	11
000 00C	12
000 00D	13
000 00E	14
000 00F	15

dezimal	sedezimal
1 000 000	0F4 240
2 000 000	1E8 480
3 000 000	2DC 6C0
4 000 000	3D0 900

100 000	018 6A0
200 000	030 D40
300 000	049 3E0
400 000	061 A80
500 000	07A 120
600 000	092 7C0
700 000	0AA E60
800 000	0C3 500
900 000	0DB BA0

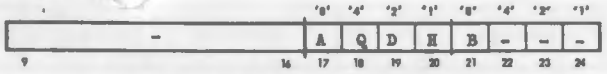
10 000	002 710
20 000	004 E20
30 000	007 530
40 000	009 C40
50 000	00C 350
60 000	00E A60
70 000	011 170
80 000	013 880
90 000	015 F90

1 000	000 3E8
2 000	000 7D0
3 000	000 BB8
4 000	000 FA0
5 000	001 388
6 000	001 770
7 000	001 B58
8 000	001 F40
9 000	002 328

100	000 064
200	000 0C8
300	000 12C
400	000 190
500	000 1F4
600	000 258
700	000 2BC
800	000 320
900	000 384

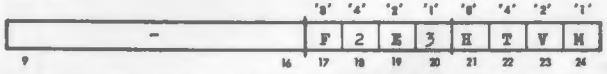
10	000 00A
20	000 014
30	000 01E
40	000 028
50	000 032
60	000 03C
70	000 046
80	000 050
90	000 05A

IR s



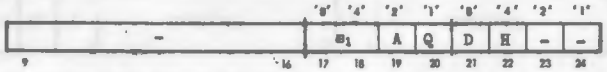
Es sind alle 2^8 möglichen Bitanordnungen erlaubt. Falls die Bits 17 - 20 = 0; Wirkung wie Nullbefehl.

LA s



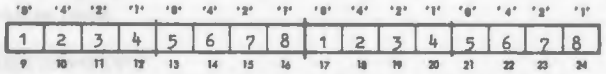
s: F, 2, E, 3, V und M können beliebig kombiniert werden. H oder T dürfen nur einzeln oder mit M verwendet werden. Nicht erlaubte Spezifikationen ergeben undefinierte Befehlsausführungen.

LR s



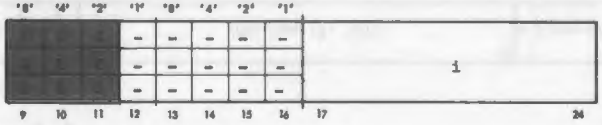
Es sind alle 2^8 möglichen Bitanordnungen erlaubt.
s_1: Typenkennung 0,1,2 oder 3

LZL s_L s_R



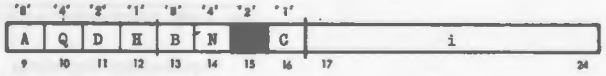
Es sind in jedem Adressenteil jeweils alle 2^8 möglichen Bitanordnungen erlaubt.

MFU, XB, M i



MFU: Bits 9 - 11 (Linksadresse) = 00L
XB : Bits 9 - 11 (Linksadresse) = 0LO
M : Bits 9 - 11 (Linksadresse) = L00

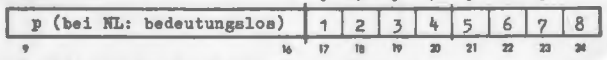
MRX, RX s i



*MRX: Bit 15 = L
RX : Bit 15 = 0
Von den Bits 9 - 13 darf höchstens eines = L sein, sonst undefinierte Befehlsausführung.

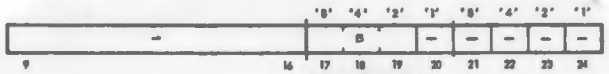
NL s

SL, SLL, SLN, SNL, SW p s



Es sind alle 2^8 möglichen Bitanordnungen erlaubt.

RM s

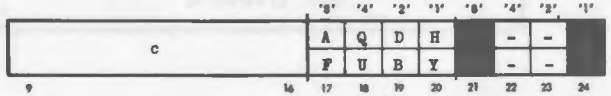


s besteht nur aus den Bits 17 - 19

** diese Bitanordnungen führen zu undefinierten Befehlsausführungen.

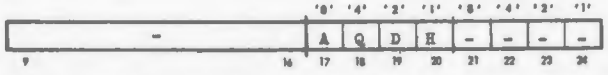
N	
L	
F	
F4	
G	
**	
FG	
**	

R, RLR c s



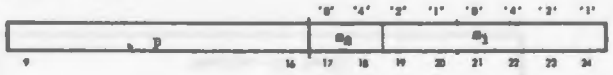
Genau eines der Bits 17 - 20 muß = L sein, sonst undefinierte Befehlsausführung. (Ausnahme: Falls c = Null, so ist Bit 17 - 20 = 0 erlaubt)
* Bit 24 = L rechte 24 Bits } nur bei Halbwortbefehlen
* Bit 24 = 0 linke 24 Bits } von Bedeutung

RT s



Genau zwei der Bits 17 - 20 müssen = L sein, sonst undefinierte Befehlsauaführung.

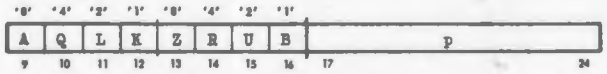
SBIT p s



s_0: Register A = Q = D = H =

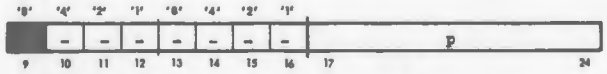
s_1: Dualziffern 1–48

SH s p



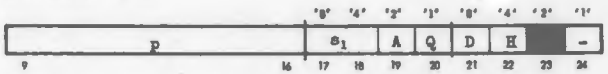
Es sind alle 2^8 möglichen Bitanordnungen erlaubt.
Wenn Bit 13 = L, sind die Bits 9 und 10 bedeutungslos.
Wenn Bit 12 = L, ist Bit 14 bedeutungslos.
Wenn Bit 11 = L und Bit 13 = 0, ist Bit 14 bedeutungslos.
Wenn Bit 9,10 und 13 = 0, so sind die Bits 11,12,14 und 15 bedeutungslos.

SHB s p



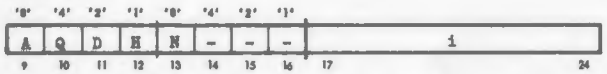
Es sind alle 2^8 möglichen Bitanordnungen erlaubt
* Bit 9 = L : links
Bit 9 = 0 : rechts

ST, STN p s



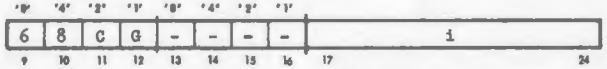
Von den Bits 19 - 22 darf nur eines = L sein, sonst erfolgt eine undefinierte Befehlsausführung. Bit 23 muß = 0 sein, sonst wird in jedem Fall gesprungen.

TRX, TXR s i



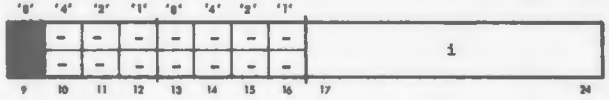
TRX: Von den Bits 9 - 12 darf nur eines = L sein, sonst undefinierte Befehlsausführungen.
TXR: Es sind alle 2^8 möglichen Bitanordnungen erlaubt.

US s i



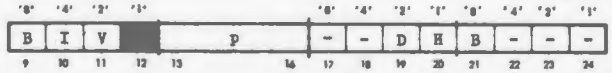
Von den Bits 9 - 11 muß genau eines = L sein, sonst undefinierbare Befehlsausführung.
Wenn das Bit 12 = 0, dann Teilspezifikation E.

XC, XCN i



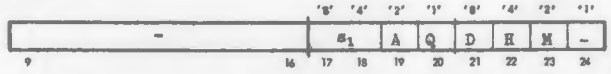
XC : Bit 9 (Linksadresse) = 0
XCN: Bit 9 (Linksadresse) = L

ZK s i



* Wenn Bit 12 = L(R), muß genau eines der Bits 19-21 = L sein sonst undefinierte Befehlsausführung
Wenn Bit 12 = 0(leer), wird in den Bits 17-24 die Adresse einer Indexzelle erwartet

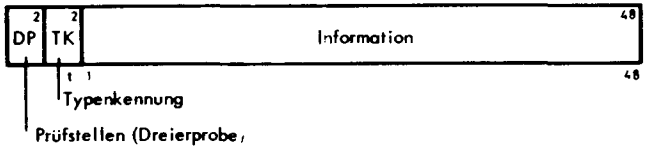
ZTR s



s_1: Typenkennung 0,1,2 oder 3
Von den Bits 19 - 22 darf höchstens eines = L sein, sonst undefinierte Befehlsausführung.

Wortstruktur (im Speicher)

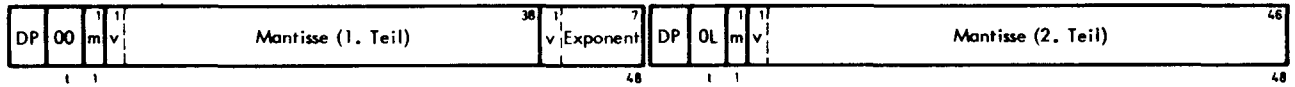
ALLGEMEIN



GLEITKOMMAZAHL (Basis 16)

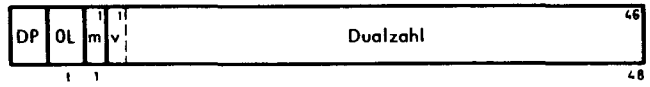
Einfache Länge

Doppelte Länge

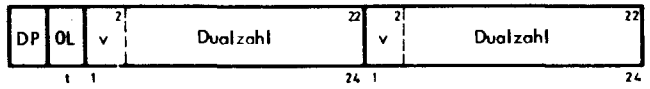


FESTKOMMAZAHL

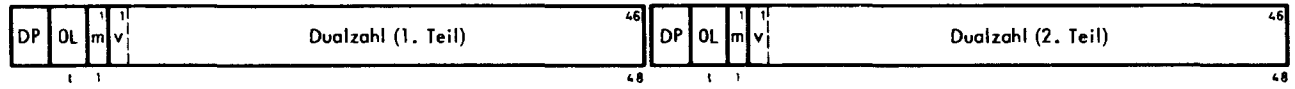
Einfache Länge



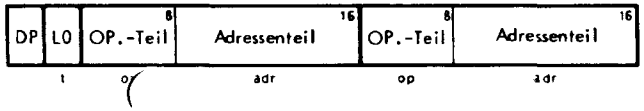
Halbe Länge (zwei Zahlen pro Wort)



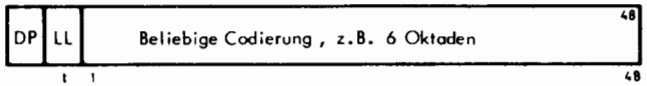
Doppelte Länge



BEFEHLE (zwei Befehle pro Wort)



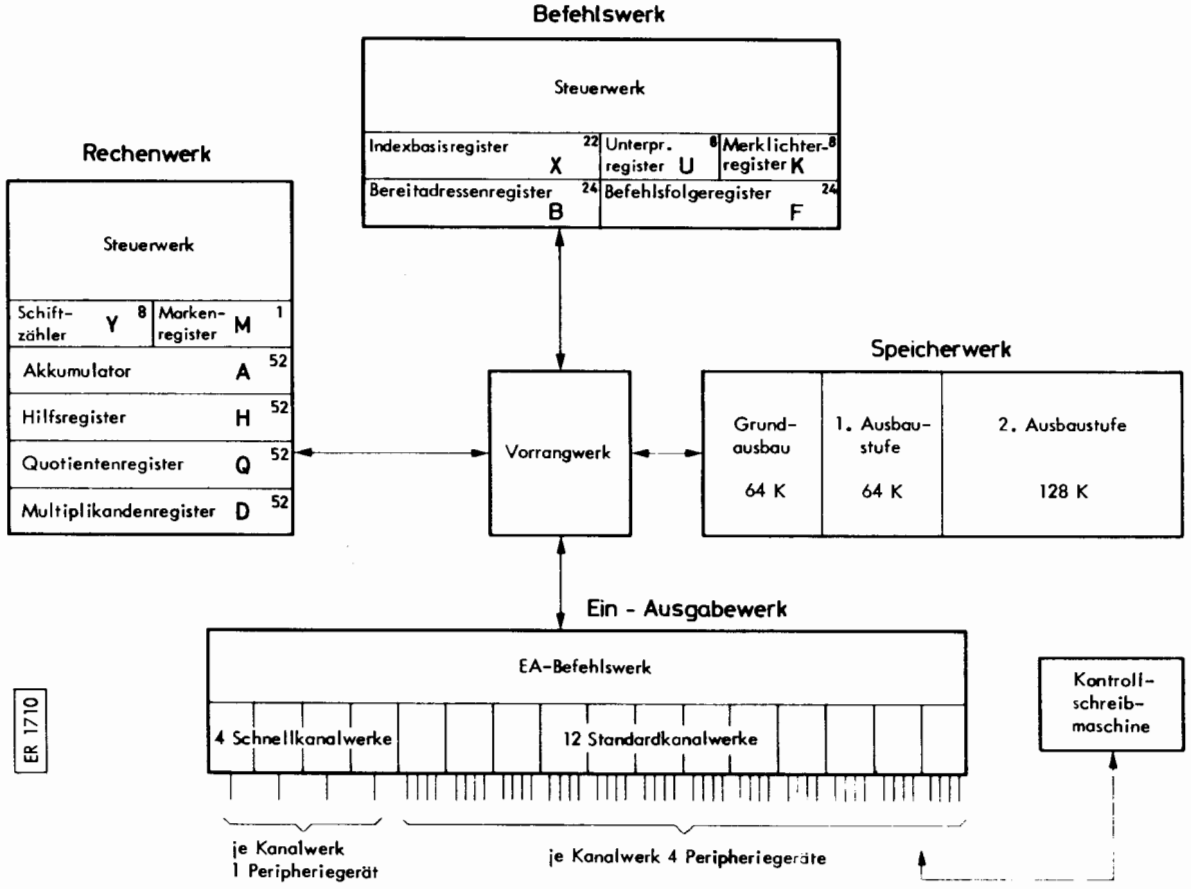
ALPHATEXT



ER 1711

DP = Bits für Prüfzwecke (Dreierprobe)
TK = Bits für Typenkennung
t = Typenkennung
m = Marke (nur im Speicher, im Register gleich der v-Stelle)
v = Vorzeichen

Blockschaltbild



ER 1710

Alphabetische Liste der Befehle

Code	Int.	Seite	mod2	R	Op
A	42	7	+	x	+
A2	7C	9	+	x	+
AA	98	7,8	+		
AB	40	7	+	x	+
AC	43	7	+		+
AQ	7E	7	+	x	+
AT	F4	9	+	x	+
ATA	89	9	+		
AU	49	8	+	x	+
AUT	69	9	+	x	+
B	70	4	+	x	+
B2	6E	4	+	x	+
B2V	6F	4	+	x	+
B2VN	67	4	+	x	+
B3	6C	4	+	x	+
B3V	6D	4	+	x	+
BA	8E	10	+		
BAN	DF	10	+		
BANR	DD	10	+		
BAR	DC	10	+		
BB	74	4	+	x	+
BC	A3	5	+		+
BCI	B6	5			+
BCL	06	6			+
BD	71	4	+	x	+
BH	73	4	+	x	+
BL	B0	5	+		
BLEI	BE	4	+		
BN	75	4	+	x	+
BNR	77	4	+	x	+
BNZ	E6	4	sp		
BQ	72	4	+	x	+
BQB	DA	4	+	x	+
BR	76	4	+	x	+
BSS	FB	4			
BT	F6	4	+	x	+
BU	D3	4	+	x	+
BZ	D9	4	+		+
BZ2	D8	4	+		+
BZN	D1	4	+		+
C	80	5	+		
C2	A0	5	+		+
C3	A1	5	+		+
CB	85	5	+		
CD	86	5	+		
CH	8F	5	+		
CMC	A2	5	+		+
CMR	83	5	+		
CMT	82	5	+		
CN	84	5	+		
CNZ	E7	5	sp		
CQ	87	5	+		
CR	81	5	+		
CT	F7	5	+		+
CU	D0	5	+		
CZ	DB	5	+		
DA	F0	8			+
DML	F2	8			+
DSB	F1	8			+
DV	60	7	+	x	+
DVD	61	7	+	x	+
DVI	62	7	+	x	+
E	29	15	sp		
EMB	28	15			
EMU	04	15	sp		
ENZ	2A	15	sp		
ET	6A	9	+	x	+
ETA	8A	9	+		
EZ	2B	15	sp		
GA	4B	8	+	x	+
GAB	52	8	+	x	+
GAC	4A	8	+		+
GDV	64	8	+	x	+
GDVI	66	8	+	x	+
GMAN	5D	8	+	x	+
GML	5E	8	+	x	+
GMLA	5F	8	+	x	+
GMLN	5C	8	+	x	+
GSB	4F	8	+	x	+
GSBB	53	8	+	x	+
GSBC	4E	8	+		+
GSBD	4C	8	+	x	+
GSBI	48	8	+	x	+
HBA	11	10			
HBC	3C	10		x	+
HBPX	0F	10			
HXP	2C	10			
HXX	2E	10			
IR	E1	16			
KDFR	94	17	+		
KFLD	95	17	+		
LA	8B	10	+		
LC	33	11	+		+
LMC	31	11	+		+
LMT	32	11	+		+
LR	9A	10	+		
LZL	10	11			
M	*08	14	sp		
M2	7A	9	+	x	+
M2N	78	9	+	x	+
M2NR	79	9	+	x	+
M2R	7B	9	+	x	+
MA	03	14	sp		
MAB	20	15			
MABI	3F	15			
MAN	5A	7	+	x	+
MANR	5B	7	+	x	+
MAR	57	7	+	x	+
MC	14	14	sp	x	+
MCE	17	14	sp	x	+
MCF	16	14	sp	x	+
MCFU	3D	14	sp	x	+
MD	09	14	sp		
MF	0B	14	sp		
MFU	*08	14	sp		
MH	2D	14			
MHX	0E	14			
ML	54	7	+	x	+
MLA	56	7	+	x	+
MLD	F3	8			+
MLN	58	7	+	x	+
MLR	55	7	+	x	+
MNA	02	14	sp		
MNR	59	7	+	x	+
MRX	*8D	14			
MU	05	15			
NL	12	11			
NRM	9F	16	+		
NULL	00	12		x	
PDP	B9	13	+		+
QBR	FE	6			+
QCR	FF	6			
R	96	15	+		
REZ	65	8	+	x	+
RLR	E0	15	sp		
RT	97	6	+		
RX	*8D	10			
S	36	12	+		
SAA	A9	13	+		
SAT	A8	13	+		
SB	46	7	+	x	+
SB2	7D	9	+	x	+
SBA	99	7,8	+		
SBB	41	7	+	x	+
SBC	47	7	+		+
SBD	45	7	+	x	+
SBI	44	7	+	x	+
SBIT	F9	13			
SBQ	7F	7	+	x	
SBT	F5	9	+	x	+
SBU	4D	8	+	x	+
SE	BC	12	sp	x	+
SEGG	93	13	+		
SFB	3A	12	+		
SFBE	FA	12	sp		
SG	AB	12	+		
SG0	D4	12	+		
SGG	AF	12	+		
SGG0	A6	12	+		
SH	9B	16	+		
SHB	21	16			
SI	AC	12	+		
SI0	A4	12	+		
SK	AA	12	+		
SK0	D5	12	+		
SKG	AE	12	+		
SKG0	A5	12	+		
SL	1E	13			
SLL	1F	13			
SLN	1C	13			
SM	34	13	+		
SMN	35	13	+		
SN	AD	12	+		
SN0	A7	12	+		
SNL	1D	13			
SR	B8	12	+		
SRN	BA	12	+		
SSR	BB	12	+		
ST	90	13	+		
STN	91	13	+		
SU	38	12	+		
SUE	BD	12	sp	x	+
SXG	CE	13			
SXGG	25	13			
SXI	24	13			
SXK	CF	13			
SXKG	26	13			
SXN	27	13			
SXR	B2	13			
SXRN	B3	13			
SZX	0A	13			
T	CC	15	sp	x	+
TBC	07	6			+
TCB	39	6	+	x	+
TDM	EA	17			+
TLD	EB	17			+
TLI	EC	17			+
TLOG	ED	17			+
TMAX	EF	17			+
TMIN	EE	17			+
TOK	FD	6			
TRX	9C	6			
TTX	0D	6			
TXR	8C	6			
TXX	0C	6			
US	E3	16	sp		
VAQ	63	16	+		
VBA	13	10			
VBC	15	10		x	+
VEL	68	9	+	x	+
VLA	88	9	+		
VXX	2F	10			
WB	F8	12			
WTR	23	6			
WTV	22	6			
XB	*08	6			
XBA	01	11			
XBAN	19	11			
XC	*18	6			
XCN	18	6			
ZI	B7	11			+
ZK	FC	6			
ZMC	30	11	+		+
ZT0	C8	11	+		+
ZT1	C9	11	+		+
ZT2	CA	11	+		+
ZT3	CB	11	+		+
ZTR	92	11	+		
ZU	3E	11	+		
ZUS	6B	9	+	x	+
ZX	1A	11			

Code: Befehlscode

Int.: Interncode in 2 Sedezimalen

mod2: + = Modifizierung 2. Art

mod2: sp = spezielle Modifizierung 2. Art

R: x = als Zweitcode beim Befehl R zugelassen

Op: Dieser Befehl holt in der Abrufphase einen Operanden aus dem Speicher

* Diese Befehle benutzen gemeinsame Interncodes.
Unterschiede liegen im Adressenteil
siehe Seiten 22/23 "Spezifikationen-Intern".

STICHWORTVERZEICHNIS

Dieses Stichwortverzeichnis beinhaltet nicht die unter Kapitel C aufgeführten EXTERN-Codes. Diese sind in Kapitel I, Seite 27 alphabetisch aufgelistet.

ABGRENZUNGSTEIL B 7,B 8
ABLAGE
-BESTIMMUNG C39
-KENNUNG C19
-SCHLUESSEL C17
-SPEZIFIKATION B 9
- BEI LITERALEN B17
- BEFEHL C19
ABSOLUTBEZUG B13,B14
ACHTELSEITENANGABE C 6
ADRESSEN
-KONSTANTE B25
-KONSTANTENSPEZIFIKATION B10
-TEIL B11,B12
-ZONEN C 8,C 9
-ZONENANORDNUNG C16
ADRESSE
- R&ENDE G23
- R&SSRFEHLER G23
ADRESSIERUNGSBEDINGUNG C10
AEND-BEFEHL C20
ALARM-BEFEHL C35
ARBEITSSPEICHER C 6
ASP
- BEFEHL C 6
- GLEICHSETZUNG C 6
ASSEMBLER A 1
AUFBEREITUNG J16,J17
AUSGABEPUFFER-ZEIGER G14

BAUSTEIN B 6
BEDINGUNGS
-LISTE D12
-OPERATOR D12
BEFEHL B11
BEFEHLSCODE B11
BENENNUNG B 7
- GLOBAL B 7
- LOKAL B 7
BENENNUNGSTEIL B 7,B 8
BEZUGSPARAMETER C22
BITFELD
-ANGABE B26
-INHALT B26
-KONSTANTE B26
-LAENGE B26
BLANK B 2
BLOCKSCHALTBILD J25
BOOLESCHE OPERATIONEN J 9

CODE
-SPEZIFIKATION C39
-UMSTEUERUNG C39
- BEFEHL C39
COMPILER A 1
CZONE-BEFEHL C11

DEF-BEFEHL D 6
DEND-BEFEHL D 6,D 7
DIFFERENZBEZUG B13
DIREKTER KERNSPEICHERBEZUG B13
DRUCK
-PARAMETER C32
- BEFEHL C32
DSP-BEFEHL C 8

EA-BEFEHLSWORT B28
EINFACHER ABSOLUTBEZUG B13
EINGABEPUFFER-ZEIGER G 7
EINGG-BEFEHL C27
ELEMENTE DER TAS-SPRACHE B 3
ENDE-BEFEHL C 6
ERSATZZEICHEN B 5
ERSETZEN J15
EXTERN
-CODE B11,J22
- BEFEHL C26
EXTOPT-BEFEHL C27

FEHLERFAELLE F 5
FESTPUNKT
-KONSTANTE B21
- ARITHMETIK J 7
FORMATFREIHEIT B 2
FORM-BEFEHL D 9
FREIHALTEANGABE C 6,C29
FZONE-BEFEHL C11

GANZ
-ADRESSE B13
-WORTLITERAL B18,F 5
-SEITENANGABE C 6
GEBAN-BEFEHL C22
GEBIETE C 8,C11
GEBIETS
-ANORDNUNG C16
-NAME C13
-PARAMETER C13,C14
- ADRESSIERUNGSBEDINGUNG C13
GEBIET-BEFEHL C13
GERADESPEZIFIKATION B 9
GLCH-BEFEHL C37
GLEICHSETZUNG C37
GLEITPUNKT
-KONSTANTE B22
- ARITHMETIK J 8
GLIEDERUNG VON QUELLENPROGRAMMEN C 4
GLOBALE MAKROVARIABLE D 5,D21
- GENV* D22
- NUMMER* D21
- VERSION* D22

HALB
-SEITENANGABE C 6
-WORTSPEZIFIKATION B10
-WORT-ARITHMETIK J 9
HDEF-BEFEHL I 3

IE-SPEZIFIKATION B 7,B 9
IGNOREZEICHEN B 2
INDEX
-ADRESSEN C29
-BENENNUNG C29
-BEZUG B15,B16
-LISTE C29
-NAME B13
-NAMENZUORDNUNG C28
-PEGEL C29
-ZELLEN C28
- ARITHMETIK J10

- BEFEHL C29
INFORMATIONSEINHEITEN B 7
- ANORDNUNG C16
INFORMATIONSTEIL B 7,B 8
INTERN
-CODE B11,J22
-SPEZIFIKATION J23

KERNSPEICHERBEZUG B13,B14
KE-BEFEHL H 8
KOMMENTAR B28
-BEGRENZUNG B28
-TEXT B28
KONSTANTE B20
KONTAKTNAMEN C25
KONTROLL
-BLOCKNAME C36
-EREIGNISSE H 8
KONVERTIERUNGSTAFEL J21

LAGERKLASSE C13
LDEF-BEFEHL I 3
LDFAB-BEFEHL I 3
LINKS
-ADRESSE B15
-SPEZIFIKATION B10
LITERAL B16
-SCHLUESSEL C17
LOESCHEN J10,J11
LUECKENNAME C16
LUECKE-BEFEHL C16

MAKRO D 6
-AUFRUF D 7,I 2
-BIBLIOTHEKSORGANISATION I 1
-DEFINITION D 6
-KONSTANTE D 5
-KONSTANTENLISTE D 5
-NAME D 6
-SPRACHE D 1
-STEUERUNG D 2
-VARIABLE D 4
-VARIABLENLISTE D 9
-VARIABLENNAME D 4
- R&ABRUCH G17
- R&AGANZ G16
- R&AGLEIT G17
- R&AOKT G15
- R&APOS G14
- R&ASKIP G15
- R&ATET G16
- R&BINAERDUMP G21
- R&DRUCK G19
- R&DRUCKE G18
- R&DRUCKETEXT G19
- R&DRUCKTEXT G19
- R&DUMP G20
- R&EBRUCH G11
- R&EGANZ G11
- R&EGLEIT G12
- R&ENDE G22
- R&EOKT G 9
- R&EPOS G 8
- R&ESKIP G 8
- R&ETET G10
- R&FEHLER G21
- R&LIES G 6
- R&RAHMEN G 4
- R&SEITE G20
- R&VORSCHUB G20
- R&WIED G12
MARKIERUNGSSPEZIFIKATION B10
MODIFIZIEREN J14,J15
MONTAGE C 1
-OBJEKT C 2
-OBJEKTNAMEN C25
MONTIERER C 2,C 3

NAMEN B 3
NOTE-BEFEHL C34

OA-BEFEHL B27
OKTADE B 6
OKTADEN
-ADRESSIERUNG B27
-FOLGE B 5
-KONSTANTE B24
-NAME B 6,C31
-WERT C31
OPERATOREN C34
OPERATORKOERPER C 2,C 3
-BESCHREIBUNG C 2,C 3
OPERATIONSTEIL B11

PARAMETER B15
PASSWORT C13
POTENZEN VON 2 J20
PROTOKOLL F 1
-RAHMEN F 1
-STEUERUNG C32
PSEUDOBEFEHL B18,B19,B20
PSEUDOBEFEHLS
-ADRESSENTEIL B18
-CODE B18

QUELLENPROGRAMM C 2,C 3
QUELLEPROTOKOLL F 2
QUERBEZUEGE C25

RECHTS
-ADRESSE B15
-SPEZIFIKATION B10
RELATIVBEZUG B13
REND-BEFEHL C39,C40
REPLIKATION C39
REPLIKATIONSBLOCK C39
REPL-BEFEHL C39

SEDEZIMALZIFFER B 4
SEGMENT C 4
-NAME C 4
SEGM-BEFEHL C 4
SEITENADRESSE C22
SELBSTDEFINIERENDER AUSDRUCK B 4
SETZEN J10,J11
SONDER
-NAMEN B 3
-ZEICHEN B 2
-ZEICHENKOMBINATIONEN B 2
SONST-BEFEHL D12,D15
SPEZIFIKATION B15,B19,B20
SPEZIFIKATIONSTEIL B 7,B 8
SPRUENGE J12,J13
STANDARDRAHMEN G 1,G 2
STARR-BEFEHL C18
START-BEFEHL C36
STEND-BEFEHL C18
STRUKT-BEFEHL C36
SYMBOLISCHE KERNSPEICHERADRESSE B13

TABELLE DURCHSUCHEN J17
TAS
- BUCHSTABE B 2
- OBJEKTE B 3
TEILSEITEN
-ANGABE C 6
-ZAHL C 6
TEILWORT-ARITHMETIK J 9
TEST
-HILFEN H 1
- BEFEHLE H 2
TETRADE B 4
TETRADEN
-FOLGE B 4
-KONSTANTE B23
TEXT
-FOLGE B 6
-KONSTANTE B25
- BEFEHL C31
TRANSPORTBEFEHLE J 4,J 5,J 6
TYPENKENNUNG B 9

UEBERSCHRIFT B28
UEBERSETZUNG C 1
UEBERSETZE-KOMMANDO E 1
UEBERWACHER H 1
UNGERADESPEZIFIKATION B 9
UNTPR-BEFEHL C35
UVB-BEFEHL C33

VEND-BEFEHL D12,D15
VERARBEITUNGSKLASSE C13
VERSION D12
VERSIONS
-KOMPLEX D12
-TEXT D12
VERS-BEFEHL D12,D13
VIERTELSEITENANGABE C 6
VORBESETZUNG C13
VORBES-BEFEHL C36
VORZEICHEN B13
VW-BEFEHL B28

WEND-BEFEHL D17,D19
WERTELISTE D17
WIEDERHOLUNG D17
WIEDERHOLUNGS
-ANGABE C39
-TEXT D17,D18
WIED-BEFEHL D17
WORTSTRUKTUR J24

XBASIS-BEFEHL C35

ZAHL B 4
ZEICHENVORRAT B 2
ZEILE-BEFEHL C33
ZENTRALCODE J19
ZIFFER B 2
ZONAN-BEFEHL C20
ZONENNAME C10
ZONE-BEFEHL C10
ZUORDNUNGSLISTE D 9
ZWEITCODE B15
ZWEITMARKIERUNGSSPEZIFIKATION B10