1 Einleitung

1.1 Kurzbeschreibung des Produkts

Der ASSEMBH ist ein Übersetzer für die Assembler- und die Makrosprache.

Der ASSEMBH bearbeitet jeweils ein Quellprogramm. Der Text eines Quellprogramms ist eine Folge von Instruktionen und Kommentaren, die jeweils eine oder mehrere Übersetzungseinheiten darstellen. Eine Übersetzungseinheit in einem Assembler-Quellprogramm fängt in der Regel mit einer START- oder CSECT-Anweisung an und endet mit einer END-Anweisung. Bei der strukturierten Programmierung (wird vom ASSEMBH-BC nicht unterstützt) werden diese Anweisungen durch die entsprechenden vordefinierten Makros generiert. Der ASSEMBH erzeugt für jede Übersetzungseinheit einen Objektmodul.

Instruktionen in einem Quellprogramm können sein:

- Assembleranweisungen
  Sie veranlassen den Assembler, während der Übersetzung bestimmte Operationen durchzuführen.
- Assemblerbefehle
  Sie veranlassen die Zentraleinheit, während des Programmlaufs bestimmte Operationen durchzuführen.
- Makroaufrufe
  Sie veranlassen den Assembler, während der Übersetzung bereits codierte Quelltexte einzulesen. Entsprechend den im Makroaufruf angegebenen Informationen werden die Quelltexte modifiziert und die so erzeugten Instruktionen ins Quellprogramm eingefügt.
- Makroanweisungen
  Sie ermöglichen es, den Text und die Anzahl der erzeugten Instruktionen entsprechend den bei der Übersetzung errechneten Bedingungen zu variieren.

Assembleranweisungen, Makroaufrufe und Makroanweisungen sowie die vordefinierten Makroaufrufe für die strukturierte Programmierung sind im folgenden beschrieben. Eine ausführliche Beschreibung der Assemblerbefehle finden Sie im Handbuch Assemblerbefehle, Beschreibung [3].

1.2 Zielgruppe

Voraussetzung für die Arbeit mit diesem Handbuch ist eine Grundausbildung in Assembler. Das Handbuch eignet sich sowohl als Nachschlagewerk, als auch dazu, neue Funktionen kennenzulernen oder Kenntnisse zu vertiefen.

1.3 Konzept des Handbuchs

Das Handbuch besteht aus drei Teilen:

- Die Kapitel 2 bis 4 beschreiben die Struktur der Assemblersprache und die Assembleranweisungen.
- Die Kapitel 5 bis 8 beschreiben die Struktur und die Elemente, sowie die Instruktionen der Makrosprache.
- Die Kapitel 9 und 10 beschreiben den Aufbau eines Assembler-Programms nach den Regeln der strukturierten Programmierung und die vordefinierten Makros für die strukturierte Programmierung.

Die Handhabung des ASSEMBH im BS2000 ist im ASSEMBH, Benutzerhandbuch [1] beschrieben. Die Beschreibung des Testsystems AID finden Sie im Handbuch AID, Testen von ASSEMBH-Programmen [2].

Literaturhinweise werden im Text in Kurztiteln angegeben. Der vollständige Titel jeder Druckschrift, auf die verwiesen wird, ist im Literaturverzeichnis aufgeführt. Zusätzlich ist dort das entsprechende Taschenbuch aufgeführt.

Hinweise zur Bestellung von Druckschriften finden Sie nach dem Literaturverzeichnis.

1.4 Änderungen gegenüber der vorigen Ausgabe

Die über das ganze Handbuch verteilten Korrekturen sind nicht eigens aufgeführt. Wesentliche fachliche Neuerungen und Änderungen sind folgende:

Die Leistung der PUNCH- und REPRO-Anweisungen wird bei Moduln im LLM-Format nicht unterstützt (Kapitel 4.2).

Folgende @-Makros im Kapitel 10 sind erweitert oder geändert:
- Operandenbeschreibung im @CONEN-Makro ist verbessert
- Erweiterung des DROP-Operanden im @END-Makro
- Operanden reg1-reg3 im @ENTR-Makro entfallen
- Textergänzung des Operanden RC im @EXIT-Makro
- Texterweiterung im @ININ-Makro
- Einführung des Ereignisnamens 'INTR' im @STXEN-Makro

Die Auflistung im Anhang 11.2: 'Format der Assemblerbefehle' wurde um die ESA-Befehle erweitert.

Die Tabelle im Anhang 11.6: 'Unterschiede von ASSEMBH zum ASSEMB' wurde erweitert und aktualisiert.

Das Kapitel "Handbuchergänzungen" enthält gesammelt weitere Neuerungen.

1.5 Metasprache

Für die Formatdarstellung der Instruktionen wird folgende Metasprache verwendet:

KONSTANTE	Großbuchstaben bezeichnen metasprachliche Konstanten, die Sie in dieser Form eingeben müssen.
name	Kleinbuchstaben bezeichnen metasprachliche Variablen, für die Sie dem Zusammenhang entsprechende Werte einsetzen müssen.
YES	Unterstreichung eines Wertes bezeichnet einen Standardwert, der vom Assembler oder vom Betriebssystem eingesetzt wird.
{YES / NO}	Geschweifte Klammern schließen Alternativen ein. Aus den angegebenen Größen müssen Sie eine auswählen. Die Alternativen stehen untereinander. Ist eine der Alternativen ein Standardwert, dann ist keine Angabe nötig, wenn Sie den Standardwert wünschen.
[ ]	Eckige Klammern schließen Wahlangaben ein, die weggelassen werden können.
( )	Runde Klammern sind metasprachliche Konstanten und müssen mit eingegeben werden.
...	Drei Punkte bedeuten, daß die vorstehende Einheit mehrmals wiederholt werden kann.
[,...]	Komma, drei Punkte bedeutet, daß die vorstehende Einheit mehrmals wiederholt werden kann, aber jeweils durch Komma getrennt werden muß. Die eckigen Klammern zeigen die Wahlfreiheit an.
{}[,...]	Die einzeilige geschweifte Klammer umschließt in diesem Fall die syntaktische Einheit, die wiederholt werden kann.
␣	Ersatzdarstellung für Leerzeichen. Wird dort verwendet, wo ein Leerzeichen syntaktisch notwendig ist.

2 Struktur der Assemblersprache

Der Text eines Assembler-Quellprogramms besteht aus einer Folge von Instruktionen und Kommentaren.
Instruktionen können hier Assembleranweisungen, Assemblerbefehle oder Makroaufrufe sein. Kommentare dienen der Programmdokumentation, sie haben keine Auswirkung auf das übersetzte Programm.
Neben diesen Assembler-Sprachelementen, die im folgenden beschrieben werden, sind im Assembler-Quellprogramm verschiedene Makro-Sprachelemente erlaubt. Diese Möglichkeit wird in Kap. 8 beschrieben.

2.1 Zeichenvorrat

Folgende Zeichen können beim Schreiben von Instruktionen verwendet werden:

Buchstaben	Ziffern	Sonderzeichen
A bis Z	0 bis 9	+ - * = , . () /
a bis z jedoch nur wenn LOW-CASE-CONVERSION=YES gesetzt ist (siehe [1])		’ (Hochkomma) & (kommerzielles Und)
_ (Unterstrich)		␣ (Leerzeichen)
$ Namen mit $ als erstem Zeichen sind für Applikationen des Betriebssystems reserviert (siehe [9])		
#		
@		

Beispiele für die Verwendung des Zeichenvorrats

Zeichen	Verwendung	Beispiel
Alphanumerische	in symbolischen Adressen	ADR100, AB_CD
Ziffern	als dezimale selbstdefinierende Werte	4096 8192
Sonderzeichen	als Operatoren:	
+	Addition	AREA1+AREA2
-	Subtraktion	OUT-20
*	Multiplikation	3*ALPHA
/	Division	NEUN/3
+ oder -	unär	+10, -4
	als Begrenzer:	
␣ Leerzeichen	zwischen Einträgen	ADR1␣LR␣R5,R6
, Komma	zwischen Operanden	OPND1,OPND2
’ Hochkomma	- um Datenkonstanten einzuschließen, - bei der Bezugnahme auf Merkmale	C’CONSTANT’ L’AREA
( ) Klammern	um Ausdrücke und Adreßkonstanten einzuschließen	MVC AREA(12), (A+B*(C-D))
	als Anzeiger für:	
. Punkt	- Folgesymbole, - Kommentar, der nicht ins Übersetzungsprotokoll eingeht - Dezimalpunkt, - Verkettungen	.LOOP .*THIS IS A COMMENT DC F’3.7C2’ &PARAM.SAVE
* Stern	- Bezugnahme auf den Adreßpegel - Kommentar, der ins Übersetzungsprotokoll eingeht	*+100 *THIS IS A COMMENT
=	- Literale, - Kennwortoperanden	MVC FELD,=C’SIEMENS’ MACALL &PAR=’ABC’
&	Variable Parameter	&PARAM

Groß- und Kleinbuchstaben im Quellprogrammtext

Der ASSEMBH erlaubt es, in Instruktionen und Kommentaren Groß- und Kleinbuchstaben zu verwenden, wenn die Option LOW-CASE-CONVERSION=YES gesetzt wurde (siehe ASSEMBH, Benutzerhandbuch [1]). Dabei ist folgendes zu beachten:

Kleinbuchstaben werden in Großbuchstaben umgeschlüsselt und entsprechend weiterverarbeitet
- in symbolischen Adressen im Namens- und Operandeneintrag,
- im Operationseintrag und
- in Assembler-Schlüsselwörtern (z.B. READ, PRVLGD,...).

Kleinbuchstaben werden unverändert verarbeitet
- im Bemerkungseintrag,
- in Kommentarzeilen,
- in C-Konstanten,
- in selbstdefinierenden Zeichenwerten und
- in Zeichenwerten der Makrosprache.

Im Übersetzungsprotokoll wird die Originalzeile mit Groß- und Kleinbuchstaben ausgegeben. Quellprogrammzeilen, die durch einen Makroaufruf generiert wurden, werden jedoch nur in Großbuchstaben ausgegeben.

Referenzlisten werden nur in Großbuchstaben ausgegeben.

2.2 Instruktionen und Kommentare

Instruktionen

Instruktionen können aus fünf Einträgen bestehen:

- dem Namenseintrag,
- dem Operationseintrag,
- dem Operandeneintrag,
- dem Bemerkungseintrag und
- dem Fortsetzungszeichen.

Diese Einträge müssen der obigen Reihenfolge entsprechen und bis auf das Fortsetzungszeichen durch mindestens ein Leerzeichen voneinander getrennt sein. Für eine Instruktion sind beliebig viele Fortsetzungszeilen erlaubt (siehe 2.7, Fortsetzungszeichen).

Die Voreinstellung für die Anfangs-, die End- und die Fortsetzungsspalte sind die Spalten 1, 71 und 16. Diese Voreinstellung kann durch eine Compiler-Option (siehe ASSEMBH, Benutzerhandbuch [1]) oder durch die ICTL-Anweisung geändert werden (siehe 4.2, ICTL-Anweisung).

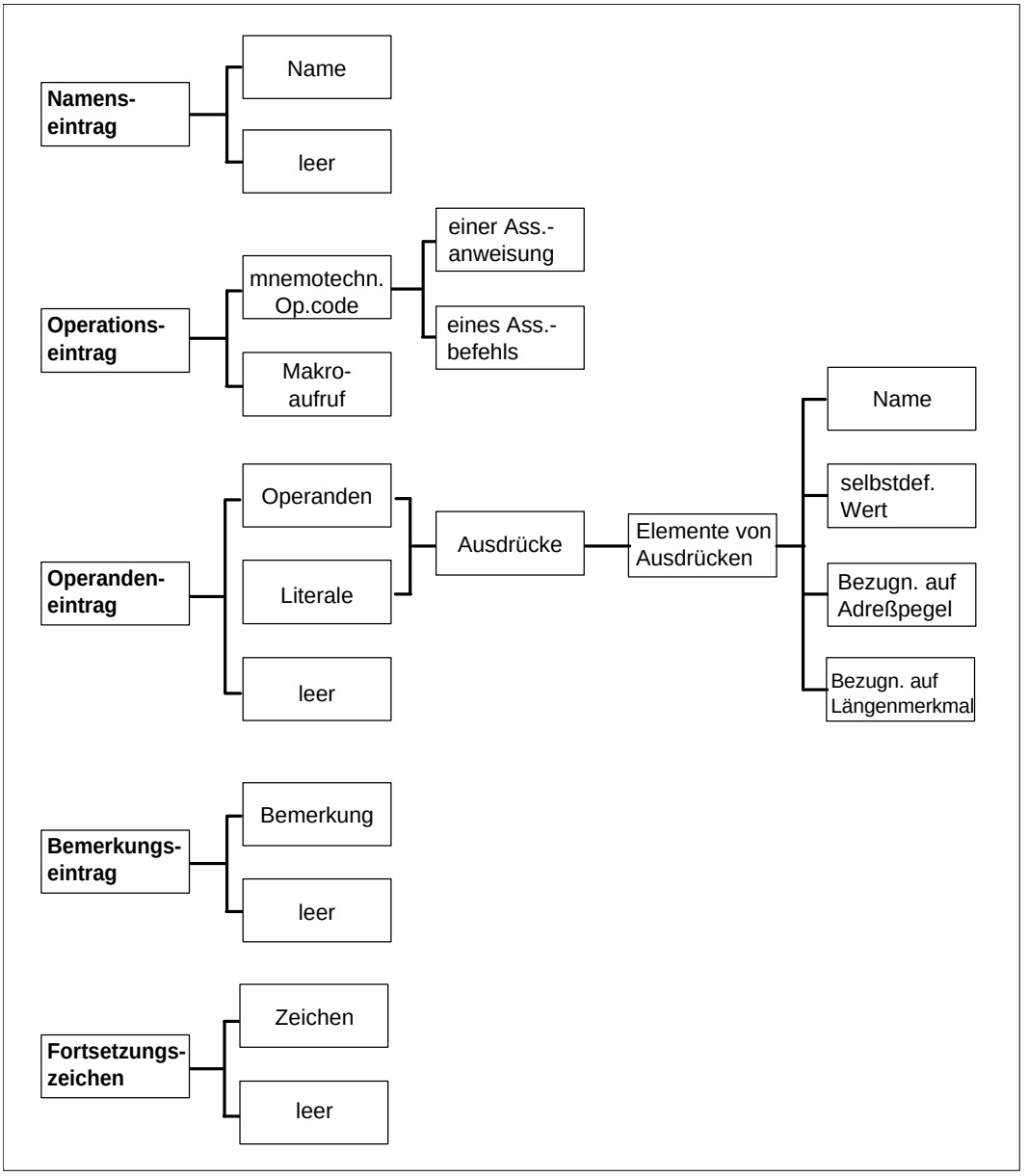


Bild 2 - 1 Aufbau von Instruktionen in der Assemblersprache

Kommentare

Kommentare werden zur Programmdokumentation verwendet. Sie haben keine Auswirkungen auf das übersetzte Programm.

Es gibt zwei Arten von Kommentarzeilen:

* Mit einem Stern in der Anfangsspalte;
Diese Kommentarzeilen werden vom Assembler übernommen und im Übersetzungsprotokoll ausgedruckt.

.* Mit einem Punkt in der Anfangsspalte, dem ein Stern folgt;
Diese Kommentarzeilen werden vom Assembler nicht übernommen und erscheinen daher auch nicht im Übersetzungsprotokoll.

Für Kommentare gibt es keine Fortsetzungszeilen. Ein trotzdem gesetztes Fortsetzungszeichen wird ignoriert. Längere Kommentare müssen mit Hilfe mehrerer Kommentarzeilen geschrieben werden.

Beispiele

```
Name        | Operation | Operand                              | Fortsetzungszeichen
------------------------------------------------------------------------------------
*  YOU NEED TWO LINES                                        | X (wird ignoriert)
*  FOR THIS COMMENT
.*  THIS COMMENT IS NOT PRINTED IN THE LISTING               |
```

2.3 Namenseintrag

Im Namenseintrag steht ein Name aus maximal 64 Buchstaben und Ziffern, der dazu dient, eine Instruktion zu identifizieren.

Externe Namen in Moduln im OM-Format (Objektmodul-Format), die vom Binder TSOSLNK bearbeitet werden, (siehe 4.2, COM-, CSECT-, DXD-, ENTRY-, EXTRN-, WXTRN, und XDSEC-Anweisung), sind auf acht Zeichen beschränkt. Längere externe Namen werden auf acht Zeichen verkürzt weiterverarbeitet und mit einer Meldung versehen.

Externe Namen in Moduln, die mit der Option COMPILER-ACT(,MODULE-FORMAT=LLM) erstellt wurden (siehe ASSEMBH, Benutzerhandbuch [1]), sind auf 32 Zeichen erweitert und werden vom Binder BINDER bearbeitet.

Der Namenseintrag kann wahlfrei sein. Wenn er vorhanden ist, muß er in der Anfangsspalte beginnen. Ist die Anfangsspalte leer, so nimmt der Assembler an, daß kein Namenseintrag vorhanden ist und interpretiert die folgenden Zeichen als Operationscode.

Der Assembler weist jedem Namen im Namenseintrag einen Adreßpegel zu. Der gleiche Name im Namenseintrag darf in einer Übersetzungseinheit nur einmal definiert sein. Namen von Programmabschnitten können mehrfach vorkommen, da Programmabschnitte unterbrochen und an anderer Stelle im Programm unter erneuter Nennung des Abschnittsnamens fortgesetzt werden können (siehe auch 4.2, CSECT-, DSECT-, AMODE- und RMODE-Anweisung).

Die Adreßpegel oder Namen sind in der Regel relativ. D.h., sie können zur Ablaufzeit einen anderen Wert haben als zur Übersetzungszeit. Wird einem Namen aber mit einer EQU-Anweisung ein absoluter Wert zugewiesen, dann ändert sich der Wert des Namens zur Ablaufzeit nicht und man spricht von einem Namen mit einem absoluten Wert.

Regeln

- Das erste Zeichen eines Namens muß ein Buchstabe sein.
- Der Wert eines Namens muß zwischen -2^{31} und $2^{31}-1$ liegen.
- Unterstriche innerhalb des Namens sind zulässig (z.B. A_B), außer in Namen, die vom Binder bearbeitet werden.
- Leerzeichen innerhalb des Namens sind nicht zulässig.
- Die Sonderzeichen & (kommerzielles Und) und . (Punkt) haben eine Sonderfunktion (siehe Kap.5, Struktur der Makrosprache).

Beispiele für zulässige Namen

```
LOOP                Field
A23456              @B4
X4F2                $A1
LOOP2               #56
LOOP_2              N
THIS_EXTREMLY_LONG_NAME_IS_REALLY_NOT_TOO_LONG_FOR_ASSEMBH
```

Beispiele für unzulässige Angaben im Namenseintrag

```
256B                (erstes Zeichen kein Buchstabe)
BCD*34              (enthält das Sonderzeichen *)
IN␣PUT              (enthält ein Leerzeichen)
BUT_NOW_THIS_EXTREMLY_GREAT_LONG_NAME_IS_TOO_LONG_EVEN_FOR_ASSEMBH
```

Definieren von Namen

Ein Name ist definiert, wenn er im Namenseintrag einer Assembleranweisung oder eines Assemblerbefehls auftritt oder als Operand einer EXTRN- oder WXTRN-Anweisung.

Die Definition von Namen beinhaltet die implizite Zuweisung eines Längenmerkmals. Das Längenmerkmal eines Namens ist die Länge des bezeichneten Speicherbereichs in Byte. So hat z.B. ein Name, der einen RX-Befehl bezeichnet, das Längenmerkmal 4.

Ausnahme

Wenn ein Name gleichgesetzt wurde mit dem Adreßpegel oder mit einem selbstdefinierenden Wert, ist das Längenmerkmal des Namens 1 (siehe 4.2, EQU-Anweisung und 5.5.8, Merkmale).

Das Längenmerkmal eines Namens wird durch einen Wiederholungsfaktor nicht beeinflußt.

2.4 Operationseintrag

Der Operationseintrag kann enthalten:

- den mnemotechnischen Operationscode einer Assembleranweisung oder eines Assemblerbefehls oder
- den Namen eines System- oder Benutzermakros (Makroaufruf).

Ein Operationseintrag muß angegeben werden. Er muß durch mindestens ein Leerzeichen vom Namenseintrag getrennt sein, bzw. bei fehlendem Namenseintrag mindestens eine Stelle rechts von der Anfangsspalte beginnen.

Ein zulässiger Operationseintrag besteht aus maximal fünf Zeichen für Assembleranweisungen und Assemblerbefehle. Falls der mnemotechnische Operationscode durch eine OPSYN-Anweisung umdefiniert wurde, sind maximal 64 Zeichen möglich. Für Makronamen sind ebenfalls maximal 64 Zeichen möglich.

Innerhalb des Operationseintrags dürfen keine Leerzeichen vorkommen.

Mit der OPSYN-Anweisung können die Standardeinstellung des Operationscodes geändert und neue mnemotechnische Operationscodes erzeugt werden (Siehe 4.2, Assembleranweisungen).

2.5 Operandeneintrag

Der Operandeneintrag enthält die Operanden, die die zu verarbeitenden Speicherbereiche, Masken, Längen oder Datenarten beschreiben oder auf sie Bezug nehmen.

Wenn ein Operandeneintrag angegeben ist, muß er durch mindestens ein Leerzeichen vom Operationseintrag getrennt sein.

Der Operandeneintrag kann aus einem oder mehreren Operanden bestehen, die wiederum einen oder mehrere Ausdrücke enthalten können. Ausdrücke bestehen ihrerseits aus Elementen und Operatoren.

Die Operanden müssen durch Kommas getrennt sein. Zwischen den Operanden und den Trennkommas dürfen keine Leerzeichen stehen.

Ein Operand darf keine Leerzeichen enthalten.

Ausnahme

Leerzeichen in Zeichenkonstanten, die als Literale, in einer Konstantendefinition oder als Direktoperanden verwendet werden.

2.5.1 Ausdrücke

Ausdrücke sind die Grundbestandteile der Operanden von Instruktionen. Sie dienen der Ermittlung eines Wertes. Ausdrücke bestehen ihrerseits aus Elementen und Operatoren. In der Assemblersprache kommen einfache und arithmetische Ausdrücke vor.

2.5.1.1 Einfache Ausdrücke

Einfache Ausdrücke bestehen aus nur einem Element. Der Wert des Ausdrucks ist gleich dem Wert des Elements.

Der Wert eines einfachen Ausdrucks muß zwischen -2^{31} und $2^{31}-1$ liegen.

2.5.1.2 Arithmetische Ausdrücke

Arithmetische Ausdrücke setzen sich zusammen aus Elementen und arithmetischen Operatoren.

Die folgenden arithmetischen Operatoren sind erlaubt:

+	Addition
–	Subtraktion
*	Multiplikation
/	Division
+	unäres Plus
–	unäres Minus

In einem arithmetischen Ausdruck kann man die Reihenfolge der Berechnung durch Klammern beeinflussen. Hierbei ist eine Schachtelung der Klammern möglich.

Regeln

– Ein arithmetischer Ausdruck darf nicht mit einem Operator, außer dem unären Plus und dem unären Minus beginnen.

  Beispiel

  richtig: `-7*(A+B)` falsch: `*A+15`

– In einem arithmetischen Ausdruck dürfen zwei Elemente nicht unmittelbar aufeinanderfolgen.

  Beispiele

  richtig: `FIELD1*(A+B)`
  `15*L'FIELD`
  falsch: `FIELD1(A+B)`
  `15L'FIELD`

– Das unäre Plus und das unäre Minus dürfen allen anderen Operatoren direkt folgen.

– Ein arithmetischer Ausdruck darf kein Literal enthalten.

– Werte von arithmetischen Ausdrücken müssen zwischen -2^{31} und $2^{31}-1$ liegen.

Beispiele

Einfache Ausdrücke	arithmetische Ausdrücke
FIELD C'FIELD' X'4040' L'FIELD *	*+32 FIELD-35 FIELD*10 FIELD/2 FIELD1+FIELD2 (OUT-(IN*L'FIELD+1)+FIELD)

Berechnung von arithmetischen Ausdrücken

Arithmetische Ausdrücke werden nach folgenden Regeln berechnet:

1. Jedem Element wird sein Wert zugeordnet.
2. Arithmetische Operationen werden von links nach rechts ausgeführt. Multiplikation und Division kommen vor Addition und Subtraktion.
3. Bei Ausdrücken, die Klammern enthalten, werden zuerst die Werte in den Klammern errechnet. Bei mehrstufigen Klammern wird zuerst der innere Klammerausdruck berechnet.
4. Jeder arithmetische Ausdruck wird auf 32 bit berechnet.
5. Divisionsreste werden vernachlässigt; daher ist das Ergebnis jeder Division ganzzahlig.

   Beispiel

   ```
   1/(2*10)       ergibt        0
   -11/2          ergibt        -5
   ```

6. Division durch Null ist zugelassen und liefert das Ergebnis Null.

2.5.1.3 Absolute und relative Ausdrücke

Ein Ausdruck wird **absolut** genannt, wenn sein Wert zur Ablaufzeit eines Programms gleich seinem Wert zur Übersetzungszeit ist.

Ein Ausdruck wird **relativ** genannt, wenn er zur Ablaufzeit eines Programms einen anderen Wert haben kann als zur Übersetzungszeit.

Ob ein Ausdruck relativ oder absolut ist, wird von den Elementen bestimmt, aus denen er gebildet ist.

Absolute Ausdrücke

Ein absoluter Ausdruck wird auf einen absoluten Wert zurückgeführt.
Ein absoluter Ausdruck kann sein:

- ein absoluter einfacher Ausdruck,

  Beispiel `L'FIELD`

- ein arithmetischer Ausdruck mit nur absoluten Elementen,

  Beispiel `L'FIELD+15`

- ein arithmetischer Ausdruck mit paarweise relativen Elementen mit entgegengesetzten Vorzeichen,

  Beispiel `*-FIELD`

- ein arithmetischer Ausdruck mit relativen und absoluten Elementen.

  Beispiel `(*-FIELD)*L'FIELD`

Mit absoluten Elementen sind alle arithmetischen Operationen erlaubt.

Wenn ein absoluter arithmetischer Ausdruck relative Elemente enthalten soll, gelten folgende Bedingungen:

- Der arithmetische Ausdruck muß eine gerade Zahl von relativen Elementen enthalten
- Die relativen Elemente müssen paarweise auftreten. D.h., die zwei relativen Elemente, die zu einem Paar gehören, müssen im gleichen Programmabschnitt definiert sein und entgegengesetzte Vorzeichen haben. Sie müssen nicht unmittelbar hintereinander stehen.
- Mit einem relativen Element darf nicht multipliziert oder dividiert werden.

Durch das **paarweise Auftreten von zwei relativen Elementen** in einem absoluten Ausdruck wird die Wirkung der Programmverschiebung zur Ablaufzeit aufgehoben. Deshalb bleibt der Wert, der von den gepaarten relativen Elementen gebildet wird, konstant, unabhängig von einer Programmverschiebung.

Beispiel

AA=AX+RY-RZ

		Wert zur Übersetzungszeit	Wert zur Ablaufzeit z.B.
AX	absolutes Element	50	50
RY	relatives Element	10	110
RZ	relatives Element	25	125
AA	absoluter Ausdruck	35	35

Relative Ausdrücke

Der Wert eines relativen Ausdrucks ändert sich um n, wenn das Programm, in dem er auftritt, zur Ablaufzeit eine um n erhöhte Ladeadresse hat.
Ein relativer Ausdruck kann sein:

– ein relativer einfacher Ausdruck,

  Beispiel `*`

– ein arithmetischer Ausdruck mit nur relativen Elementen,

  Beispiel `*-FIELD1-TERM1`

– ein arithmetischer Ausdruck mit relativen und absoluten Elementen.

  Beispiel `*-FIELD1-TERM1+L'FIELD`

Wenn ein relativer arithmetischer Ausdruck nur relative oder relative und absolute Elemente enthalten soll, gelten folgende Bedingungen:

– erste Möglichkeit: Der arithmetische Ausdruck enthält eine ungerade Anzahl von relativen Elementen und diese Elemente treten, bis auf eines, paarweise auf (siehe oben).

  Beispiel `*-FIELD1-TERM1`

  * und FIELD1 sollen paarweise auftreten, d.h., sie sind im gleichen Programmabschnitt definiert und haben entgegengesetzte Vorzeichen.

– zweite Möglichkeit: Mehrere relative Elemente sind im gleichen Programmabschnitt definiert und haben keine entgegensetzten Vorzeichen. Dann muß das Ergebnis ihrer Berechnung ebenfalls im gleichen Programmabschnitt liegen.
  Der arithmetische Ausdruck kann zusätzlich zu diesen einzelnen relativen Elementen gepaarte relative Elemente enthalten.

  Beispiel `TERM1+TERM2+TERM3+*-FIELD1`

  TERM1,TERM2 und TERM3 sind im gleichen Programmabschnitt definiert, treten aber nicht paarweise auf. Dann muß das Ergebnis ihrer Berechnung ebenfalls im gleichen Programmabschnitt liegen.

– Mit keinem der relativen Elemente darf multipliziert oder dividiert werden.

Ein relativer Ausdruck wird auf einen relativen Wert zurückgeführt. Dieser wird errechnet aus dem Wert der ungepaarten relativen Elemente, verändert um den Wert der absoluten und gepaarten relativen Elemente.

Beispiel

RA=RU-RV+RU-10

		Wert zur Übersetzungszeit	Wert zur Ablaufzeit z.B.
RU	relatives Element	10	110
RV	relatives Element	5	105
10	absolutes Element	10	10
RU-RV	gepaarte rel. Elemente	5	5
RA	relativer Ausdruck	5	105

2.5.2 Elemente von Ausdrücken

Ein Element eines Ausdrucks ist die kleinste Einheit der Assemblersprache, die einen Wert darstellt.

Jedem Element entspricht ein Wert. Dieser Wert wird entweder durch den Assembler zugewiesen (Namen, Längenmerkmale, Adreßpegel) oder er ist in dem Element direkt enthalten (selbstdefinierender Wert).

Jedes Element kann allein einen einfachen Ausdruck darstellen oder mit anderen zu einem arithmetischen Ausdruck kombiniert werden.

Ein Element wird **absolut** genannt, wenn sein Wert unabhängig von der Ladeadresse des Programms ist, d.h., daß sein Wert zur Ablaufzeit gleich seinem Wert zur Übersetzungszeit ist.

Ein Element wird **relativ** genannt, wenn es relativ zum Programmanfang ist, d.h., daß es zur Ablaufzeit einen anderen Wert haben kann als zur Übersetzungszeit.

Wenn zwei relative Elemente in einem Ausdruck kombiniert werden, können sie **paarweise** auftreten. Das bedeutet, daß diese zwei Elemente im gleichen Programmabschnitt definiert sein müssen und entgegengesetzte Vorzeichen haben.

Folgende Elemente von Ausdrücken treten in der Assemblersprache auf:

- Namen (absolut oder relativ)
- Selbstdefinierende Werte (absolut)
- Bezugnahmen auf den Adreßpegel (relativ)
- Bezugnahmen auf das Längenmerkmal (absolut)

Im folgenden werden die Elemente von Ausdrücken und die Regeln für ihre Verwendung erörtert.

2.5.2.1 Namen

Für Namen im Operandeneintrag gelten dieselben Aussagen, die bereits im Abschnitt 2.3 für Namen im Namenseintrag gemacht wurden.

2.5.2.2 Selbstdefinierende Werte

Selbstdefinierende Werte dienen zur direkten Darstellung von Werten. Sie werden benutzt, um Daten, Masken, Register und Adreßerhöhungen anzugeben. Selbstdefinierende Werte dürfen maximal ein Wort lang sein.

Selbstdefinierende Werte sind absolute Elemente, da ihr Wert zur Ablaufzeit gleich ihrem Wert zur Übersetzungszeit ist.

Zu den selbstdefinierenden Werten gehören:

- dezimale,
- sedezimale,
- binäre und
- Zeichenwerte

Beispiele

Selbstdefinierender Wert	Dezimal-Wert	Binär-Wert	
241	241	1111 0001	dezimal
X'F1'	241	1111 0001	sedezimal
X'101'	257	1 0000 0001	sedezimal
B'1111'	15	0000 1111	binär
B'11110001'	241	1111 0001	binär
C'1'	241	1111 0001	Zeichenwert
C'A'	193	1100 0001	Zeichenwert

Selbstdefinierende Werte, Konstanten, Literale

Die Anwendung selbstdefinierender Werte unterscheidet sich von der Anwendung der Datenkonstanten und der Literale:

- Wenn Datenkonstanten oder Literale im Operanden einer Instruktion spezifiziert werden, übernimmt der Assembler ihre Adressen in die Instruktion.
- Wenn dagegen ein selbstdefinierender Wert in einer Instruktion verwendet wird, übernimmt der Assembler seinen Wert in die Instruktion.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------
FIELD     | DS        | CL3
CONST     | DC        | C'ABC'
*         |           |
          | MVI       | FIELD,X'FF'        (01)
          | MVC       | FIELD,CONST        (02)
          | MVC       | FIELD,=C'ABC'      (03)
```

(01) Der Wert des Direktoperanden soll nach FIELD übertragen werden. Der Assembler übernimmt FF in den generierten Maschinenbefehl.

(02) Der Inhalt von CONST soll nach FIELD übertragen werden. Der Assembler übernimmt die Adresse von CONST in den generierten Maschinenbefehl.

(03) Das Literal C'ABC' soll nach FIELD übertragen werden. Der Assembler übernimmt die Adresse des Literals in den generierten Maschinenbefehl.

Dezimale selbstdefinierende Werte

Ein dezimaler selbstdefinierender Wert ist eine vorzeichenlose Dezimalzahl, dargestellt durch eine Folge von Dezimalziffern, in der auch führende Nullen vorkommen können.

Ein dezimaler selbstdefinierender Wert wird vom Assembler in sein binäres Äquivalent übersetzt.

Begrenzungen des Wertes hängen jeweils vom Verwendungszweck ab:

- Ein dezimaler selbstdefinierender Wert darf in keinem Fall aus mehr als zehn Ziffern bestehen oder den Wert 2^{31}-1 überschreiten.
- Ein dezimaler selbstdefinierender Wert, der ein Mehrzweckregister bezeichnen soll, muß einen Wert von 0 bis 15 haben;
- Ein Wert, der eine Distanz darstellt, darf die höchste Arbeitsspeicheradresse nicht überschreiten.

Sedezimale selbstdefinierende Werte

Ein sedezimaler selbstdefinierender Wert ist eine vorzeichenlose Sedezimalzahl, dargestellt als Folge von Sedezimalziffern. Die Ziffernfolge muß in Hochkommas eingeschlossen sein und unmittelbar dem Buchstaben X folgen.

Jede Sedezimalziffer wird vom Assembler in einen Binärwert umgesetzt, der 4 Bits belegt. D.h., daß z.B. ein sedezimaler selbstdefinierender Wert, der eine Acht-bit-Maske darstellen soll, aus zwei Sedezimalziffern besteht.

Der größte zugelassene Sedezimalwert ist X'FFFFFFFF'.

Binäre selbstdefinierende Werte

Ein binärer selbstdefinierender Wert ist eine vorzeichenlose Folge von Binärziffern, die zur Darstellung beliebiger Binärmuster verwendet werden. Die Ziffernfolge muß in Hochkommas eingeschlossen sein und unmittelbar auf den Buchstaben B folgen.

Ein binärer selbstdefinierender Wert kann bis zu 32 Binärziffern enthalten. Der Wert wird auf ganze Bytes aufgefüllt, dazu wird das höchstwertige Byte links mit binären Nullen aufgefüllt.

Binäre selbstdefinierende Werte werden hauptsächlich zur Darstellung der Bitmuster von Masken oder in logischen Befehlen verwendet.

Beispiel

```
Name    | Operation | Operand
--------|-----------|-----------------
MASK    | EQU       | B'10101101'
ALPHA   | TM        | GAMMA,MASK
```

Der binäre selbstdefinierende Wert dient als Maske in einem TM-Befehl. Der Inhalt von GAMMA soll Bit für Bit mit dem Bitmuster der Maske verglichen werden.

Selbstdefinierende Zeichenwerte

Mit einem selbstdefinierenden Zeichenwert kann man abdruckbare Zeichen darstellen. Ein selbstdefinierender Zeichenwert besteht aus ein bis vier Zeichen, die in Hochkommas eingeschlossen sind. Unmittelbar vorhergehen kann der Buchstabe C.

In selbstdefinierenden Zeichenwerten können alle Buchstaben, Ziffern und Sonderzeichen verwendet werden.

Bei der Verwendung von Hochkomma (') und kommerziellem Und (&) gilt:
Das Hochkomma (') wird in der Assemblersprache und das kommerzielle Und (&) in der Makrosprache als syntaktisches Zeichen verwendet. Deshalb müssen für jedes Hochkomma oder jedes &-Zeichen, die in einem selbstdefinierenden Zeichenwert verwendet werden sollen, zwei Hochkommas bzw. zwei &-Zeichen geschrieben werden. Die beiden Hochkommas bzw. &-Zeichen werden als einzelnes Hochkomma, bzw. als einzelnes & übersetzt.

Beispiele

`A'$`	muß geschrieben werden als	`C'A''$'`
`'A'`	muß geschrieben werden als	`C'''A'''`

2.5.2.3 Bezugnahme auf den Adreßpegel

Durch die Verwendung eines Sterns (*) als Element eines Ausdrucks kann man an jeder Stelle des Quellprogramms den aktuellen Adreßpegel ansprechen.

Dem Stern wird der Wert des aktuellen Adreßpegels zugewiesen, d.h., der Adreßpegel der Instruktion, in der der Stern verwendet wird.

Die Bezugnahme auf den Adreßpegel kann in allen Assemblerbefehlen verwendet werden und in den Assembleranweisungen DC, DS, EQU, ORG und USING, die den Stern im Operandeneintrag erlauben. Außerdem kann man sich in Adreßkonstanten auf den Adreßpegel beziehen und in Literalen, die als Adreßkonstanten spezifiziert sind (Zur Verwendung des Stern in Adreßkonstanten mit Wiederholungsfaktor siehe DC-Anweisung, 4.2). Ein Sonderfall ist die Verwendung des Sterns in Format 2 der USING-Anweisung (siehe 4.2). Hier bezeichnet er die Anfangsadresse des Speicherbereichs für den Pseudoregistervektor.

Der Maximalwert des Adreßpegels ist 2^{24}-1. Das entspricht einer Modulgröße von 16 MB.

Beispiele

```
Adreßpegel      Quellprogrammanweisung                  Wert von *
(sedezimal)

LOCTN           SOURCE STATEMENT

000100          NAME     B      *+8                     ┐
000104                   B      NAME+8                  ├>NAME (=000100)
000108  <─── Zieladresse beider Sprungbef.              ┘

000120          CONSTANT DC     A(*)                    CONSTANT (=000120)

000134          ALPHA    L      R5,=A(*)                ALPHA (=000134)
```

2.5.2.4 Bezugnahme auf das Längenmerkmal

Auf das Längenmerkmal eines Namens bezieht man sich, indem man unmittelbar vor dem Namen ein L’ einsetzt. Der Assembler ersetzt diesen Ausdruck durch die implizite Länge des Namens.

Das Längenmerkmal von * (L’*) ist gleich der Länge der Instruktion, in der die Bezugnahme vorkommt. Eine Ausnahme ist die Anweisung EQU * ohne Längenoperand (siehe 4.2, EQU-Anweisung). Hier ist das Längenmerkmal 1.

Beispiel

Im Beispiel wird unter Verwendung des Längenmerkmals eine Zeichenkonstante an die Stelle der höchst- bzw. niedrigstwertigen Bytes eines Feldes gebracht.

```
Name     | Operation | Operand
---------|-----------|-------------------------
A1       | DS        | CL8                      (01)
B2       | DC        | CL2’AB’                  (02)
*        |           |
HIORD    | MVC       | A1(L’B2),B2              (03)
LOORD    | MVC       | A1+L’A1-L’B2(L’B2),B2    (04)
```

(01) A1 bezeichnet einen Speicherbereich von 8 byte Länge und hat das Längenmerkmal 8.

(02) B2 kennzeichnet eine Zeichenkonstante von 2 byte und hat das Längenmerkmal 2.

(03) Der Befehl mit der Adresse HIORD überträgt den Inhalt von B2 in die zwei höchstwertigen Bytes von A1. Der Wert L’B2 gibt die benötigte Länge an. Bei der Übersetzung des Befehls wird die Länge in das entsprechende Feld der Instruktion gebracht.

(04) Der Befehl mit der Adresse LOORD überträgt den Inhalt von B2 in die beiden niedrigstwertigen Bytes von A1. Der arithmetische Ausdruck A1+L’A1-L’B2 ergibt als Adresse das siebente Byte von A1. Zur Anfangsadresse von A1 wird die Länge von A1 addiert und davon die Länge von B2 subtrahiert. Der Inhalt von B2 wird also in das siebte und achte Byte vom Feld A1 übertragen. Auch hier liefert L’B2 die für den Befehl benötigte Längenangabe.

2.5.3 Literale

Mit Literalen können Daten, z.B. Zahlenwerte, Adressen oder abdruckbare Zeichen, definiert werden, ohne daß dafür DC-Anweisungen angegeben werden. Literale können den zweiten Operanden von Assemblerbefehlen ersetzen.

Jedem Literal wird vom Assembler Speicherplatz im Literalbereich zugewiesen. Im generierten Assemblerbefehlscode wird für das Literal die Adresse im Literalbereich eingetragen.

Format eines Literals: $=[dup]typ[Ln_1][Sn_2][En_3]\left\{\begin{array}{l}'chrcon' \\ 'datcon[,...]' \\ (adrcon[,...])\end{array}\right.$

dup — Wiederholungsfaktor
dezimaler selbstdefinierender Wert oder positiver absoluter Ausdruck in Klammern,
Wertebereich: 0 bis 2^{24}-1

typ — Literaltyp
ein einzelner Buchstabe, kann für Zeichenkonstanten entfallen

mögliche Literaltypen:

C	Zeichenkonstante
B	binäre Konstante
P, Z	dezimale Konstanten
X	sedezimale Konstanten
F, H	Festpunktkonstanten
E, D, L	Gleitpunktkonstanten
A, Y, V	Adreßkonstanten

Ln_1 — Längenfaktor;
n_1 ist ein dezimaler selbstdefinierender Wert oder ein positiver absoluter Ausdruck in Klammern

Sn_2 — Skalenfaktor

En_3 — Exponentenfaktor
n_2, bzw. n_3 ist ein positiver oder negativer dezimaler selbstdefinierender Wert oder ein absoluter Ausdruck in Klammern

chrcon — Wert der Zeichenkonstanten

datcon — Wert der dezimalen, sedezimalen, binären, Festpunkt- und Gleitpunktkonstanten

adrcon — Wert der Adreßkonstanten

Regeln

- Ein Literal beginnt mit einem Gleichheitszeichen (=). Für die Schreibweise eines Literals gelten die gleichen Regeln wie für einen Operanden der DC-Anweisung (siehe 4.2, insbesondere Modifizierfaktoren und Konstantentypen).
- Literale können in allen Assemblerbefehlen eingesetzt werden, bei denen der Operand ein relativer Ausdruck ist, der nur für Lesezugriffe verwendet wird.
  In Assemblerbefehlen, die als Operand einen relativen Ausdruck enthalten, der für Schreibzugriffe verwendet wird, dürfen Literale nicht verwendet werden.
- Literale dürfen nicht in Assembleranweisungen benutzt werden.
- Ein Literal darf nicht Element eines Ausdrucks sein.
- Der Wiederholungsfaktor in einem Literal darf nicht Null sein.
- Adreßkonstanten vom Typ Q und S dürfen nicht in Literalen verwendet werden.

Hinweis

Der Typ eines in einem Assemblerbefehl angegebenen Literals wird nicht auf Übereinstimmung mit dem Operationscode des Befehls überprüft.

Literalbereich

Der Assembler bestimmt den Wert der Literale und legt sie in einem bestimmten Bereich des Speichers, dem Literalbereich, ab.

Standardmäßig liegt der Literalbereich am Ende des ersten Programmabschnitts. Im Übersetzungsprotokoll wird der Literalbereich in diesem Fall nach der END-Anweisung protokolliert. Mit der Assembleranweisung LTORG (siehe 4.2) können mehrere Literalbereiche definiert und ihre Lage im Programm beliebig gewählt werden.

Gleiche Literale werden im Literalbereich nur einmal abgelegt, außer es handelt sich um Konstanten mit Bezug auf den Adreßpegel.

Beispiele

```
Name      | Operation | Operanden
----------+-----------+-------------------------
AREA1     | DS        | 3CL4                  } diese Definitionen gelten
AREA2     | DS        | PL3                   } für alle folgenden
HW        | DS        | H                     } Befehle
          | .         |
          | .         |
          | MVC       | AREA1,=3CL4'ABCD'       (01)
          | UNPK      | AREA2,=P'352'           (02)
          | MVC       | HW,=H'80'               (03)
          | IC        | 5,=X'FF'                (04)
          | LM        | 4,6,=A(AREA1,HW,*)      (05)
```

(01) In das Feld AREA1 wird übertragen: C1C2C3C4C1C2C3C4C1C2C3C4.
(02) AREA2 enthält den entpackten Wert F3F5C2.
(03) In das Feld HW wird der Wert 0050 übertragen.
(04) Der IC-Befehl lädt den Wert FF in das niedrigstwertige Byte von Register 5.
(05) Die Adressen von AREA1 und HW und der aktuelle Wert des Adreßpegels werden in die Register 4,5, und 6 geladen.

2.6 Bemerkungseintrag

Der Bemerkungseintrag enthält Erläuterungen zum Programm, die im Übersetzungsprotokoll enthalten sein sollen.

In Bemerkungen dürfen alle zulässigen Zeichen, einschließlich Leerzeichen, verwendet werden (siehe 2.1, Zeichenvorrat).

Der Bemerkungseintrag muß durch mindestens ein Leerzeichen vom Operandeneintrag getrennt sein.
Soll der Eintrag in einer weiteren Zeile fortgesetzt werden, muß in der Fortsetzungszeichenspalte ein Fortsetzungszeichen stehen.

In Instruktionen, die keinen Operanden, jedoch Bemerkungen enthalten sollen, muß vor dem Bemerkungseintrag ein Leerzeichen-Komma-Leerzeichen stehen.

Beispiel

```
Name        | Operation | Operand
------------|-----------|-----------------------
            | CSECT     | ␣,␣COMMENT
            | .         |
            | .         |
            | MVC       | FIELD1,FIELD2␣COMMENT
            | .         |
            | .         |
            | END       | ␣,␣COMMENT
```

2.7 Fortsetzungszeichen

Die erste Spalte nach der Endspalte ist die Fortsetzungszeichenspalte, in dieser steht das Fortsetzungszeichen. Die Voreinstellung für die Endspalte ist Spalte 71. Das Fortsetzungszeichen muß gesetzt werden, wenn eine Instruktion in einer weiteren Zeile fortgesetzt werden soll. Dazu kann man jedes Zeichen, außer dem Leerzeichen, verwenden.

Vor dem Fortsetzungszeichen muß kein Leerzeichen stehen.

Enthält die Fortsetzungszeichenspalte ein Leerzeichen, wird angenommen, daß die Instruktion abgeschlossen ist.

Die Instruktion muß in der Fortsetzungsspalte der nächsten Zeile fortgesetzt werden:

- Beginnt der Text in der Fortsetzungsspalte mit einem Zeichen ungleich dem Leerzeichen, wird er als Fortsetzung des entsprechenden Eintrags interpretiert.
- Steht in der Fortsetzungsspalte ein Leerzeichen, dann wird der folgende Text als nächster Eintrag gewertet.

Für eine Instruktion sind beliebig viele Fortsetzungszeilen erlaubt.

3 Adressierung, Programmunterteilung und Programmverknüpfung

3.1 Adressierung

Die Adresse kennzeichnet einen Speicherplatz. Sie besteht aus dem Inhalt eines Basisadreßregisters und der Distanz, die dazu addiert wird (Basis/Distanz-Adressierung). Im Fall von RX-Befehlen kann zusätzlich der Inhalt eines Indexregisters addiert werden.

Für die Angabe von Adressen in einem Quellprogramm gibt es zwei Möglichkeiten:

Nicht-symbolische Adressen

In diesem Fall werden Basisadreßregister und Distanz direkt bezeichnet (siehe Assemblerbefehle, Beschreibung [3]).

Symbolische Adressen

Eine symbolische Adresse wird erst vom Assembler in die Basis/Distanz-Form umgesetzt.
Eine symbolische Adresse ist ein Name oder wird über variable Parameter bzw. eine Verkettung von variablen Parametern und alphanumerischen Zeichen erzeugt.

Verwendung von symbolischen Adressen

Damit symbolische Adressen vom Assembler in ihre nicht-symbolische Form umgesetzt werden können, muß der Programmierer

- dem Assembler mit einer USING-Anweisung angeben, welche Mehrzweckregister als Basisadreßregister benutzt werden,
- in der USING-Anweisung angeben, welcher Wert jedem dieser Register als Basisadresse zuzuweisen ist und
- zur Programmlaufzeit jedes der Basisadreßregister mit dem angegebenen Wert laden.

Bei der Übersetzung wird die symbolische Adresse vom Assembler in ihre nicht-symbolische Form umgesetzt und anschließend in den Objektcode übersetzt.

Wenn symbolische Adressen verwendet werden, muß ein Programm für jeden Programmabschnitt eine eigene USING-Anweisung enthalten.

Mit der USING-Anweisung zugewiesene Basisadreßregister können mit Hilfe der DROP-Anweisung wieder für andere Zwecke freigegeben werden.

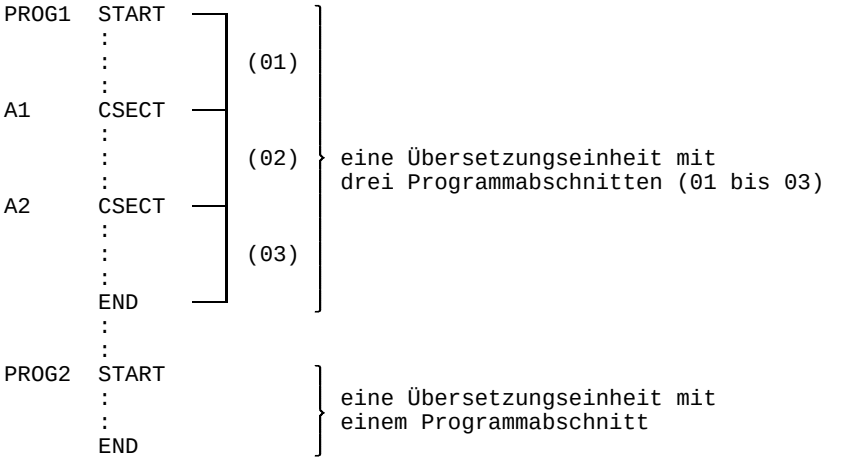
3.2 Programmunterteilung

Der Text eines Assembler-Quellprogramms besteht aus einer oder mehreren Übersetzungseinheiten. Eine Übersetzungseinheit fängt in der Regel mit einer START- oder CSECT-Anweisung an und wird durch eine END-Anweisung abgeschlossen. Sie wird meist als "ein Programm" bezeichnet.
Jede Übersetzungseinheit wird in einen Objektmodul übersetzt.

Eine Übersetzungseinheit kann aus einem oder mehreren Programmabschnitten bestehen, die als Teile eines Objektmoduls übersetzt werden.

Die entsprechenden Anweisungen an den Binder ermöglichen es, einen oder mehrere Objektmodule zu einem ablauffähigen Programm zu binden.

Beispiel



Es ist im allgemeinen notwendig, sich auf Daten zu beziehen, die in einem anderen Programmteil definiert sind oder in einen anderen Programmteil verzweigen. Zu diesem Zweck muß man die Kommunikation zwischen den Programmteilen ermöglichen.

- Jeder einzelne Programmabschnitt, der angesprochen wird, muß symbolisch adressierbar sein (siehe 3.1, Symbolische Adressen)
- Die zwei oder mehr Übersetzungseinheiten, die verbunden werden sollen, müssen symbolisch verknüpft werden (siehe 3.4, symbolische Programmverknüpfung).

3.3 Programmabschnitte

Ein Programmabschnitt ist der kleinste Programmteil, der unabhängig von den anderen beim Laden verschoben werden kann, ohne die Ablauflogik des Programms zu beeinflussen. Eine Übersetzungseinheit kann maximal 2^{15}-1 Programmabschnitte enthalten.

Es gibt ausführbare Programmabschnitte, die in den Objektcode übersetzt werden und Referenz-Programmabschnitte, die nicht in den Objektcode übersetzt werden. Sie werden benutzt, um Daten zu beschreiben, auf die man sich von ausführbaren Programmabschnitten aus beziehen kann.

Adreßpegel in Programmabschnitten

Teile verschiedener Programmabschnitte können in einer Übersetzungseinheit in vermischter Reihenfolge auftreten, da der Assembler für jeden Programmabschnitt einen eigenen Adreßpegel führt.

Der Adreßpegel von Referenzprogrammabschnitten hat den Anfangswert Null, d.h., daß die Adreßwerte innerhalb eines Programmabschnitts relativ zum Anfang des Programmabschnitts sind.

Den ausführbaren Programmabschnitten wird aufeinanderfolgender Speicherplatz zugewiesen, und zwar in der Reihenfolge, in der sie im Quellprogrammtext auftreten. Jeder nach dem ersten folgende Programmabschnitt beginnt an der nächsten freien Doppelwortgrenze, außer, wenn das Merkmal PAGE angegeben wurde (siehe 3.3.3, Merkmale von Programmabschnitten).

Der Maximalwert des Adreßpegels in einem Programmabschnitt, bzw. der Maximalwert des Gesamtadreßpegels aller Programmabschnitte ist 2^{24}-1.

3.3.1 Ausführbare Programmabschnitte

Ein ausführbarer Programmabschnitt (**C**ode-**Sect**ion) wird durch eine START- oder CSECT-Anweisung eingeleitet und in den Objektcode übersetzt.

Erster Programmabschnitt

Der erste Programmabschnitt in einer Übersetzungseinheit kann durch die START-Anweisung gekennzeichnet werden, mit der eine vorläufige Startadresse des Programms angegeben werden kann (siehe 4.2, START-Anweisung).

Am Ende des ersten Programmabschnitts wird vom Assembler der Literalbereich angelegt, wenn dessen Lage nicht durch die LTORG-Anweisung anders festgelegt wurde.

Falls der erste Programmabschnitt an anderer Stelle fortgesetzt werden soll, muß dazu eine CSECT-Anweisung mit dem gleichen Namenseintrag verwendet werden.

Weitere ausführbare Programmabschnitte

Sollen in einer Übersetzungseinheit nach dem ersten weitere ausführbare Programmabschnitte auftreten, müssen diese mit einer CSECT-Anweisung eingeleitet werden. Diese Programmabschnitte können jeweils mit einer CSECT-Anweisung mit gleichem Namen an einer anderen Stelle fortgesetzt werden (siehe 4.2, CSECT-Anweisung).

3.3.2 Referenz-Programmabschnitte

Für Referenz-Programmabschnitte wird kein Objektcode erzeugt. Sie werden verwendet, um Daten zu beschreiben, die von ausführbaren Programmabschnitten angesprochen werden sollen. Sie werden durch eine DSECT-, XDSEC-, COM- oder DXD-Anweisung eingeleitet.

Pseudoabschnitt

Ein Pseudoabschnitt (**D**ummy-**Sect**ion) beschreibt eine Datenstruktur, die wie eine "Schablone" auf einen Speicherbereich projiziert werden soll. Eine DSECT-Anweisung kennzeichnet den Beginn dieser Datenstruktur. Die Namen, die in einem Pseudoabschnitt definiert sind, entsprechen den Strukturelementen.

Die Projektion der Datenstruktur auf einen Speicherbereich geschieht in folgender Weise:

- Die Adresse des gewünschten Speicherbereichs muß während des Programmlaufs in ein Register geladen werden.
- Dieses Register muß im Programm mit einer USING-Anweisung als Basisadreßregister für den Pseudoabsabschnitt definiert werden. Dadurch wird die Datenstruktur immer auf den Speicherbereich projiziert, dessen Adresse im Basisadreßregister geladen ist.

Werden nun in Assemblerbefehlen Namen verwendet, die in einem Pseudoabschnitt definiert sind, dann werden sie beim Programmlauf von den Daten aus dem Speicherbereich ersetzt, auf den die Datenstruktur projiziert wurde.

Externer Pseudoabschnitt

Ein externer Pseudoabschnitt wird durch eine XDSEC-Anweisung gekennzeichnet. Für seine Verwendung gelten die gleichen Regeln wie für einen Pseudoabschnitt (siehe dort), bis auf folgende Ausnahme:

Für die Definition eines externen Pseudoabschnitts (XDSEC D) werden Externinformationen an den Binder übergeben, mit denen dieser die Externinformationen, die für Referenzen eines externen Pseudoabschnitts (XDSEC R) in einer anderen Übersetzungseinheit angegeben werden, zu befriedigen sucht. Der Adreßpegel wird in der Referenz eines externen Pseudoabschnitts (XDSEC R) auf Null gesetzt und bleibt Null für den gesamten Abschnitt. Der wirkliche Adreßpegel eines Namens in einer Referenz eines externen Pseudoabschnitts wird vom Binder in den Befehl, in dem dieser Name benutzt wird, eingetragen.

Somit ist die Auswertung der Adreßpegel für Namen einer XDSEC R von der Assemblerebene auf die Binderebene verlagert. Dies hat den Vorteil, daß die Änderung eines Symbols einer XDSEC D nicht eine Neuübersetzung sämtlicher Module, die diese XDSEC benutzen, nötig macht, sondern nur der Module, die den zu ändernden Namen benutzen.

Hinweis

Zur Zeit unterstützt der DLL (bis BS2000 V9.5) die Verwendung von externen Pseudoabschnitten nicht, d.h., daß Programme mit TSOSLNK gebunden werden müssen und kein Nachlademechanismus in Anspruch genommen werden kann.
Ab BS2000 V10.0 unterstützt der DBL die Verwendung von externen Pseudoabschnitten sowie das dynamische Nachladen.

Gemeinsamer Hilfsabschnitt

Ein gemeinsamer Hilfsabschnitt ist ein Speicherbereich, der von mehreren unabhängig voneinander übersetzten Übersetzungseinheiten aus angesprochen werden kann, die gebunden und als ein Programm geladen wurden. Er wird durch eine COM-Anweisung gekennzeichnet.

Ein gemeinsamer Hilfsabschnitt kann durch DS- und DC-Anweisungen in Teilfelder zerlegt werden, die relativ zu seinem Anfang definiert sind. D.h., die Struktur des gemeinsamen Hilfsabschnitts ist bereits vorgegeben. Konstanten, die mit DC-Anweisungen in einem gemeinsamen Hilfsabschnitt definiert wurden, werden nicht übersetzt. Die Felder können erst beim Programmlauf mit Werten versorgt werden.

Ein gemeinsamer Hilfsabschnitt muß in jeder Übersetzungseinheit adressierbar sein, von der aus er angesprochen werden soll. Damit eine Kommunikation zwischen Übersetzungseinheiten mit einem gemeinsamen Hilfsabschnitt möglich ist, muß er in jeder identisch definiert sein.

Pseudoregister

Pseudoregister sind Speicherbereiche, die von mehreren Übersetzungseinheiten aus angesprochen werden können. Sie dienen als Arbeitsbereiche und der Kommunikation verschiedener Übersetzungseinheiten untereinander. Die Pseudoregister können an beliebiger Stelle in einer Übersetzungseinheit definiert werden und müssen nicht in aufsteigender und/oder zusammenhängender Reihenfolge erscheinen.

Der Assembler berechnet Ausrichtung und Länge der Pseudoregister und gibt diese Informationen über ESD-Einträge an den Binder weiter. Während des Bindevorgangs werden alle Pseudoregister aus allen zusammenzubindenden Moduln zu einem "Pseudoregistervektor" zusammengefaßt. Hierbei wird vom Binder Ausrichtung, Länge und Ablage der einzelnen Pseudoregister bestimmt.
Beim Binden werden die jeweils stärksten Attribute bei gleichen Pseudoregistern als letztendlich gültige Attribute verwendet. Wäre z.B. ein Pseudoregister in einer CSECT auf Halbwortgrenze, in einer anderen auf Wortgrenze ausgerichtet, so wäre dieses Pseudoregister nach dem Binden auf Wortgrenze ausgerichtet.

Der vom Binder zusammengestellte Pseudoregistervektor definiert nur die Struktur der Pseudoregister. Der Speicherbereich dafür muß explizit in einem ausführbaren Programmabschnitt bereitgestellt werden.

Die Länge dieses Pseudoregistervektors kann vom Binder in ein Wort eingetragen werden, das mit der CXD-Anweisung für diesen Zweck reserviert werden muß.

Die Verwendung von Pseudoregistern hat gegenüber dem Einsatz von gemeinsamen Hilfsabschnitten und externen Pseudoabschnitten folgende Vorteile:

1. Gegeben sei CSECT1 mit PSREG1 und PSREG2
   CSECT2 mit PSREG3 und PSREG4.

   Beide CSECTs sollen zu einem Programm zusammengebunden werden.

   Man kann nun innerhalb einer CSECT nur auf die Pseudoregister zugreifen, die in eben dieser CSECT definiert sind. In CSECT1 also nur auf PSREG1 und PSREG2. Ein absichtliches oder versehentliches Überschreiben von PSREG3 und PSREG4 ist nicht möglich.

2. Ändert man in einem Modul die Anzahl oder die Datenbeschreibung der Pseudoregister, so braucht man nur diesen Modul neu zu übersetzen und das gesamte Programm neu zu binden. Ändert man dagegen die Datenbeschreibung eines gemeinsamen Hilfsabschnitts oder eines externen Pseudoabschnitts, so müssen alle Moduln, die von einer Veränderung seines Inhalts oder seiner Länge betroffen sind, geändert und neu übersetzt werden.

Hinweis

Zur Zeit unterstützt der DLL (bis BS2000 V9.5) die Verwendung von Pseudoregistern nicht, d.h., daß Programme mit TSOSLNK gebunden werden müssen und kein Nachlademechanismus in Anspruch genommen werden kann.
Ab BS2000 V10.0 unterstützt der BINDER die Verwendung von Pseudoregistern sowie das dynamische Nachladen.

Die Adressierung der Pseudoregister kann auf zwei Arten erfolgen:

- über ein Basisadreßregister
  Die Anweisung USING *PRV,8 (siehe 4.2, USING-Anweisung, Format 2 und Anhang 11.4, Beispiel 2) bewirkt z.B., daß für alle Adressen, die Pseudoregister betreffen, das Register 8 als Basisregister verwendet wird, unabhängig vom sonst gültigen Basisregister. Register 8 muß dann noch mit der Anfangsadresse des Speicherbereichs für den Pseudoregistervektor geladen werden.
- über Q-Konstanten (siehe Anhang 11.4, Beispiel 1)
  Hierbei steht zur Programmlaufzeit in der Q-Konstanten der Offset des Pseudoregisters zum Anfang des Pseudoregistervektors. D.h., daß dieser Wert zum Anfang des Speicherbereichs für den Pseudoregistervektor addiert werden muß, um das Pseudoregister ansprechen zu können.

Pseudoregister können auf zweierlei Weise definiert werden (siehe Anhang 11.4, Beispiele):

- Mit der DXD-Anweisung;
- Mit der DSECT-Anweisung unter Verwendung von Q-Konstanten;
  hierdurch wird der gesamte Pseudoabschnitt als Pseudoregister definiert. Die DSECT-Struktur kann für Zugriffe innerhalb dieses Pseudoregisters verwendet werden.

3.3.3 Merkmale von Programmabschnitten

Um das dynamische Binden und Laden zu ermöglichen, ist es häufig notwendig, allen Daten und Instruktionen innerhalb eines Programmabschnitts bestimmte Eigenschaften mitzugeben. Die folgenden Merkmale können (in beliebiger Reihenfolge) im Operandenfeld von START-, CSECT- und COM-Anweisungen angegeben werden:

PUBLIC — Dieses Merkmal bedeutet, daß der Abschnitt gemeinschaftliche Daten oder Instruktionen (mehrfach benutzbar) enthält.

READ — Dieses Merkmal bedeutet, daß der Abschnitt nicht modifiziert werden kann, d.h., schreibgeschützt ist (read only).

PRVLGD — Dieses Merkmal bedeutet, daß dem Abschnitt ein Sicherheitsschlüssel zugewiesen wird, so daß nur privilegierte Systemroutinen Zugriff zu ihm haben.

PAGE — Dieses Merkmal zeigt an, daß der Abschnitt an einer Seitengrenze beginnen soll, die ein Vielfaches von 4096 ist.

RESIDENT — Dieses Merkmal zeigt an, daß der angegebene Abschnitt in den Speicher geladen und dort resident gehalten wird.

Merkmale können einzeln oder zu mehreren spezifiziert werden. Der endgültige Vorrat von Eigenschaften eines Programmabschnitts wird von der Kombination aller Merkmale bestimmt. Die Merkmale können in der Anweisung stehen, die den Beginn des Programmabschnittes definiert. Es ist nicht notwendig, Merkmale bei Anweisungen mit gleichem Namen zu wiederholen, es sei denn, aus Dokumentationsgründen.

Wenn keine Merkmale angegeben sind, wird der Programmabschnitt als privat, modifizierbar und an Doppelwortgrenze ausgerichtet betrachtet. Im Objektmodul sind für jeden Programmabschnitt Informationen über seine Eigenschaften abgelegt.

Adressierungsmodus und Ladeattribut

Diese zwei Merkmale werden einem Programmabschnitt mit der AMODE- bzw. RMODE-Anweisung zugeordnet.

AMODE — Diese Anweisung ordnet einem Programmabschnitt einen Software-Adressierungsmodus zu, der seinerseits den Hardware-Adressierungsmodus bezeichnet, den der Programmabschnitt bei seiner Ausführung erwartet.

RMODE — Diese Anweisung ordnet einem Programmabschnitt ein Ladeattribut zu. Dieses gibt an, in welchen Bereich des Adreßraumes (oberhalb oder unterhalb von 16 Megabyte) das Programm geladen werden muß, bzw. kann.

3.4 Symbolische Programmverknüpfung

Die symbolische Programmverknüpfung ermöglicht es, symbolische Adressen, die in einer Übersetzungseinheit definiert sind, von einer anderen aus anzusprechen. Der Assembler benötigt dafür entsprechende Informationen, die er über ESD-Einträge an den Binder weitergibt. Der Binder ersetzt diese symbolischen Bezugnahmen vor bzw. beim Laden durch aktuelle Adressen.

Eine symbolische Adresse, die von einer anderen Übersetzungseinheit aus angesprochen werden soll, muß dem Assembler und dem Binder durch die ENTRY-Anweisung kenntlich gemacht werden. Sie ist damit als symbolische Adresse einer Einsprungstelle definiert.

Wenn in einer Übersetzungseinheit symbolische Adressen verwendet werden, die in einer anderen Übersetzungseinheit definiert sind, müssen sie durch die EXTRN- oder WXTRN-Anweisung kenntlich gemacht werden. Da die Definition solcher symbolischen Adressen in einer anderen Übersetzungseinheit erfolgt, weist der Assembler vorläufig einen Wert 0 und ein Längenmerkmal 1 zu.
In der Übersetzungseinheit, die die EXTRN-Adresse verwendet, muß ein Basisregister für Zugriffe auf diese Adresse bereitgestellt werden. Der Wert der Adresse muß über eine A-Konstante in das Basisregister geladen werden (siehe 4.2, DC-Anweisung).

Eine weitere Möglichkeit zur symbolischen Verknüpfung ist die Verwendung von V-Konstanten (siehe 4.2, DC-Anweisung). Diese Konstanten werden als indirekte Verknüpfungspunkte betrachtet, die aus einer extern definierten symbolischen Adresse erzeugt werden. In diesem Fall darf die symbolische Adresse nicht mit der EXTRN-Anweisung gekennzeichnet sein.
V-Konstanten können für Sprünge in andere Übersetzungseinheiten verwendet werden; jedoch nicht für Bezugnahmen auf Daten in anderen Übersetzungseinheiten.

Ein Beispiel für die Verknüpfung von zwei unabhängigen Übersetzungseinheiten ist in der Beschreibung der EXTRN-Anweisung (siehe 4.2) zu finden.

4 Assembleranweisungen

4.1 Allgemeines

Anders als Assemblerbefehle, die die Zentraleinheit veranlassen, während des Programmlaufs bestimmte Operationen auszuführen, veranlassen Assembleranweisungen den Assembler, während der Übersetzung bestimmte Operationen durchzuführen. Sie dienen der Steuerung des Übersetzungsvorgangs und übernehmen Hilfsfunktionen.

Programmierhinweise

1. In Assembleranweisungen dürfen keine Literale als Operanden verwendet werden.
2. Werden Assembleranweisungen als Modellanweisungen in einer Makrodefinition verwendet, können Namens-, Operations- und Operandeneintrag mit Hilfe von **variablen Parametern** generiert werden (siehe 5.1.2, Aufbau der Makrodefinition und Kap. 6, variable Parameter).
3. Bei einigen Anweisungen ist im Namenseintrag die Angabe eines Folgesymbols erlaubt. Ein Folgesymbol im Namenseintrag kennzeichnet die Anweisung als Sprungziel im Rahmen der Makrosprache (siehe 5.3.1, Folgesymbole). Ein Folgesymbol im Nameneintrag kann nicht mit variablen Parametern generiert werden.

Die Assembleranweisungen lassen sich in folgende Gruppen einteilen:

Zuweisen von Werten und Eigenschaften

EQU	Gleichsetzen
OPSYN	Mnemotechnischen Operationscode zuweisen

Definition von Datenbereichen

DC	Konstante definieren
DS	Speicherbereich reservieren
CXD	Speicherplatz für die Länge des Pseudoregistervektors reservieren

Basisregisteranweisungen

USING	Basisadreßregister zuweisen
DROP	Basisadreßregister freigeben
STACK	USING- oder PRINT-Status sichern
UNSTK	USING- oder PRINT-Status restaurieren

Programmunterteilung, Programmverknüpfung, Merkmale von Programmabschnitten

START	Programmanfang definieren
CSECT	Programmabschnitt definieren
DSECT	Pseudoabschnitt definieren
XDSEC	Externen Pseudoabschnitt definieren
COM	Gemeinsamen Hilfsabschnitt definieren
DXD	Pseudoregister definieren
END	Übersetzungsende
AMODE	Adressierungsmodus zuordnen
RMODE	Ladeattribut zuordnen
ENTRY	ENTRY-Adresse kennzeichnen
EXTRN	EXTRN-Adresse kennzeichnen
WXTRN	Bedingte EXTRN-Adresse kennzeichnen

Steuerung der Eingabe

ICTL	Eingabeformat steuern
COPY	Quellprogrammtext aus Bibliothekselement kopieren

Ausgabe in den Objektmodul

ORG	Adreßpegel setzen
LTORG	Literalbereich definieren
CNOP	Nulloperation setzen
PUNCH	Text in Objektmodul kopieren
REPRO	Folgezeile in Objektmodul kopieren

Steuerung der Protokollierung

Die Anweisungen zur Steuerung der Protokollierung kennzeichnen die Einzelheiten des Übersetzungsprotokolls. Sie wirken sich nur auf das Übersetzungsprotokoll aus und erzeugen keine Befehle oder Konstanten im Quellprogramm.

TITLE	Protokollüberschrift
SPACE	Zeilenvorschub
EJECT	Seitenvorschub
PRINT	Protokollinhalt steuern
STACK	USING- oder PRINT-Status sichern
UNSTK	USING- oder PRINT-Status restaurieren

Hinweis

Die Anzahl der Zeilen pro Seite auf dem Übersetzungsprotokoll kann nicht mit einer dieser Anweisungen beeinflußt werden. Sie muß mit der entsprechenden Option gesteuert werden (siehe ASSEMBH, Benutzerhandbuch [1]).

4.2 Beschreibung der Anweisungen

AMODE Adressierungsmodus zuordnen

Funktion

Die AMODE-Anweisung ordnet einem Programmabschnitt einen Adressierungsmodus zu.

Format

Name	Operation	Operanden
[{name / .sym}]	AMODE	{24 / 31 / ANY}

name Name
.sym Folgesymbol

Beschreibung

name bezieht sich auf einen gleichnamigen Programmabschnitt und muß mit dem Namen einer START-, CSECT- oder COM-Anweisung übereinstimmen.

Bei leerem Namensfeld bezieht sich die AMODE-Anweisung auf einen unbenannten Programmabschnitt.

.sym Ein Folgesymbol ist gleichbedeutend mit einem leeren Namensfeld.

24 Dem Programmabschnitt wird der 24-bit-Adressierungsmodus zugeordnet.

31 Dem Programmabschnitt wird der 31-bit-Adressierungsmodus zugeordnet.

ANY Der Programmabschnitt ist sowohl im 24-bit- als auch im 31-bit-Adressierungsmodus ablauffähig.

Zu ’Adressierungsmodus’ siehe Abschnitt 3.3.3 und das Handbuch Einführung in die XS-Programmierung [7].

Die Information über den Adressierungsmodus eines Programmabschnitts wird im ESD-Eintrag ausgegeben.

Programmierhinweise

1. Die AMODE-Anweisung kann an beliebiger Stelle im Quellprogramm stehen. Das Quellprogramm darf beliebig viele AMODE-Anweisungen enthalten, wobei ein spezifizierter Namen nur einmal auftreten darf.
2. Für einen unbenannten gemeinsamen Hilfsabschnitt (siehe 4.2, COM-Anweisung) darf keine AMODE-Anweisung gesetzt werden.
3. Der Adressierungsmodus, der einem Programmabschnitt zugeordnet wird, überträgt sich auch auf die ENTRY-Namen dieses Programmabschnitts.
4. Falls keine Spezialfunktionen erbracht werden sollen, sollten einem Programmabschnitt die Attribute AMODE = ANY und RMODE = ANY zugewiesen werden.

Kombinationen von AMODE und RMODE

Wenn für einen Programmabschnitt die AMODE-Anweisung gesetzt wird, sind die folgenden Kombinationen mit der RMODE-Anweisung für den gleichen Programmabschnitt möglich:

AMODE 24 mit RMODE 24
AMODE 31 mit RMODE 24 oder RMODE ANY
AMODE ANY mit RMODE 24 oder RMODE ANY

Voreinstellung

Folgende Voreinstellungen gelten, wenn für einen Programmabschnitt entweder AMODE oder RMODE oder beide nicht gesetzt wurden:

angegeben	Voreinstellung
weder AMODE noch RMODE	} AMODE 24 und RMODE 24
AMODE 24	RMODE 24
AMODE 31	RMODE 24
AMODE ANY	RMODE 24
RMODE 24	AMODE 24
RMODE ANY	AMODE 31

Tabelle 4-1 Voreinstellung für AMODE, bzw. RMODE

>>>>> siehe auch RMODE-Anweisung

CNOP Nulloperation setzen

Funktion

Mit der CNOP-Anweisung kann der Adreßpegel für die folgende Instruktion auf bestimmte Bytes in einem Wort oder Doppelwort ausgerichtet werden.

Format

Name	Operation	Operanden
[{name / .sym}]	CNOP	b,w

name Name
.sym Folgesymbol
b absoluter Ausdruck, mögliche Werte: 0,2,4,6
w absoluter Ausdruck, mögliche Werte: 4,8

Beschreibung

b gibt an, auf welches Byte in einem Wort oder Doppelwort der Adreßpegel gesetzt werden soll

w gibt an, ob das in b angegebene Byte in einem Wort (w=4) oder Doppelwort (w=8) liegen soll.

Nachfolgende Tabelle zeigt die möglichen Kombinationen von b und w und ihre Bedeutung.

b,w	Bedeutung
0,4	Beginn eines Wortes
2,4	Mitte eines Wortes (zweites Halbwort)
0,8	Beginn eines Doppelwortes
2,8	Zweites Halbwort eines Doppelwortes
4,8	Mitte (drittes Halbwort bzw. zweites Wort) eines Doppelwortes
6,8	Viertes Halbwort eines Doppelwortes

Tabelle 4-2 Zulässige Kombinationen der Operanden in der CNOP-Anweisung

Die nächste Tabelle zeigt die Stelle eines Wortes, bzw. Doppelwortes, die jedes dieser Operandenpaare bezeichnet.

Doppelwort							
Wort				Wort			
Halbwort		Halbwort		Halbwort		Halbwort	
Byte	Byte	Byte	Byte	Byte	Byte	Byte	Byte
0,4 0,8		2,4 2,8		0,4 4,8		2,4 6,8	

Tabelle 4-3 Ausrichtung des Adreßpegels mit der CNOP-Anweisung

Programmierhinweise

1. Wenn die verlangte Ausrichtung eine Erhöhung des Adreßpegels erfordert, dann werden durch die CNOP-Anweisung ein bis drei Nulloperationen (siehe BCR und NOPR in Assemblerbefehle, Beschreibung [3]) erzeugt, die je zwei Bytes belegen.
2. Ist der Adreßpegel bereits entsprechend ausgerichtet, dann werden durch die CNOP-Anweisung keine Nulloperationen erzeugt.
3. Der Name einer CNOP-Anweisung erhält als Wert den Adreßpegel **vor** einer evtl. notwendigen Erhöhung durch Nulloperationen. Die CNOP-Anweisung selbst wird auf Halbwortgrenze ausgerichtet.

COM Gemeinsamen Hilfsabschnitt definieren

Funktion

Die COM-Anweisung kennzeichnet den Beginn oder die Fortsetzung eines gemeinsamen Hilfsabschnitts, der von mehreren Übersetzungseinheiten aus angesprochen werden kann, und reserviert Speicherplatz für ihn.

Format

Name	Operation	Operanden
[{name / .sym}]	COM	[typ[,...]]

name Name
.sym Folgesymbol
typ Merkmalkennzeichnung für Programmabschnitte (siehe 3.3.3, Merkmale von Programmabschnitten)

Beschreibung

Der Namenseintrag kennzeichnet den Namen des gemeinsamen Hilfsabschnitts.

Eine COM-Anweisung ohne Namen kennzeichnet einen unbenannten gemeinsamen Hilfsabschnitt.
Der Beginn eines unbenannten COM-Abschnitts wird im ESD- und XREF-Listing protokolliert (siehe ASSEMBH, Benutzerhandbuch [1]).

Die Anfangsadresse eines gemeinsamen Hilfsabschnitts ist immer auf Doppelwortgrenze ausgerichtet.

Ein gemeinsamer Hilfsabschnitt erhält einen eigenen Adreßpegel mit dem Anfangswert Null.

Programmierhinweise

1. Ein gemeinsamer Hilfsabschnitt kann durch DS- oder DC-Anweisungen in Teilfelder zerlegt werden, die dann relativ zu seinem Beginn definiert sind.
2. Assemblerbefehle oder Konstanten, die in einem gemeinsamen Hilfsabschnitt angegeben sind, werden nicht übersetzt. Die Felder eines gemeinsamen Hilfsabschnitts können nur beim Programmablauf versorgt werden (siehe Beispiel).
3. Innerhalb eines Programms können mehrere COM-Anweisungen mit demselben Namen auftreten. Die erste bezeichnet den Beginn, die weiteren die Fortsetzung des gemeinsamen Hilfsabschnitts.
4. Ein gemeinsamer Hilfsabschnitt, der von 2 Übersetzungseinheiten aus angesprochen werden soll, muß in jeder der beiden identisch definiert und adressierbar sein.
5. Für einen unbenannten Hilfsabschnitt darf keine AMODE- und keine RMODE-Anweisung gesetzt werden.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------------
PR0G      | START     |
R2        | EQU       | 2
          | .         |
          | .         |
          | L         | R2,=A(HCOM)          (01)
          | USING     | HCOM,R2              (02)
          | MVC       | COM2,=C'12345'       (03)
          | .         |
          | .         |
HCOM      | COM       |                      ⎫
COM1      | DS        | F                    ⎬ (04)
COM2      | DS        | CL5                  ⎭
          | .         |
          | .         |
```

(01) Lädt die Anfangsadresse von HCOM ins Register R2.
(02) Gibt HCOM als Basisadresse für den gemeinsamen Hilfsabschnitt an und weist Register R2 als sein Basisadreßregister zu.
(03) Versorgt das Feld COM2 mit Daten.
(04) Definition des gemeinsamen Hilfsabschnitts.

>>>>> siehe auch END-, START-, CSECT-, DSECT- und XDSEC-Anweisung

COPY Quellprogrammtext aus Bibliothekselement kopieren

Funktion

Die COPY-Anweisung kopiert das genannte Element aus einer Bibliothek in ein Quellprogramm.

Format

Name	Operation	Operanden
	COPY	name

Beschreibung

name ist der Name des zu kopierenden Bibliothekselementes.

Die COPY-Anweisung prüft nicht, ob name bezüglich der Bibliotheksverwaltung syntaktisch zulässig ist.

Die kopierten Anweisungen des Bibliothekselements werden hinter der COPY-Anweisung in das Quellprogramm eingefügt. Über eine Benutzersteuerung kann angegeben werden, in welcher Bibliothek, bzw. in welcher Abteilung einer Bibliothek gesucht werden soll (siehe LMS, Benutzerhandbuch [4] und ASSEMBH, Benutzerhandbuch [1]).

Existieren in einer Bibliothek mehrere Elemente mit gleichem Namen, dann wird das Element mit der höchsten Versionsbezeichnung verwendet.

Programmierhinweise

1. Die Schachtelungstiefe von COPY-Anweisungen kann über eine Option der //COMPILE-Anweisung angegeben werden (siehe ASSEMBH, Benutzerhandbuch [1]). Sie ist auf 5 voreingestellt und kann maximal 255 betragen.
2. COPY-Anweisungen können sowohl im Assembler-Quellprogramm als auch innerhalb von Makros stehen.
   COPY-Anweisungen innerhalb von inneren Makrodefinitionen werden erst bei der Definitionsbearbeitung des inneren Makros expandiert.
   Kopierte Anweisungen können ihrerseits Makrodefinitionen enthalten.
3. Der Name eines COPY-Elementes darf nicht generiert werden.
4. Der kopierte Text wird gemäß einer eventuell vorausgegangenen ICTL-Anweisung interpretiert. Da dies unbeabsichtigt sein kann, wird eine Warnung ausgegeben.

5. Der kopierte Text darf keine ICTL-Anweisung enthalten.
6. Das Protokollieren der kopierten Anweisungen kann über eine Option der //COMPILE-Anweisung (siehe ASSEMBH, Benutzerhandbuch [1]) oder mit PRINT COPY bzw. PRINT NOCOPY (siehe 4.2, PRINT-Anweisung) gesteuert werden.

CSECT Programmabschnitt definieren

Funktion

Die CSECT-Anweisung kennzeichnet den Beginn oder die Fortsetzung eines Programmabschnitts.

Format

Name	Operation	Operanden
[{name / .sym}]	CSECT	[typ[,...]]

name Name
.sym Folgesymbol
typ Merkmalkennzeichnung für Programmabschnitte (siehe 3.3.3, Merkmale von Programmabschnitten)

Beschreibung

Der Namenseintrag kennzeichnet den Namen des Programmabschnitts, der mit der CSECT-Anweisung beginnt oder fortgesetzt wird.

Eine CSECT-Anweisung ohne Namen kennzeichnet einen unbenannten Programmabschnitt.
Der Beginn einer unbenannten CSECT wird im ESD- und XREF-Listing protokolliert (siehe ASSEMBH, Benutzerhandbuch [1]).

Das Längenmerkmal von name ist 1.

Programmierhinweise

1. Alle Instruktionen, die zwischen einer CSECT-Anweisung und der nächsten CSECT- bzw. DSECT-Anweisung mit anderen Namen stehen, gehören zu einem Programmabschnitt.
2. Innerhalb eines Programms können mehrere CSECT-Anweisungen mit demselben Namen auftreten. Die erste bezeichnet den Beginn, die weiteren die Fortsetzung des Programmabschnitts. Eine zweite bzw. weitere CSECT-Anweisung ohne Namen kennzeichnet die Fortsetzung eines unbenannten Programmabschnitts.
3. Soll ein Assembler-Quellprogramm mit AID getestet werden, muß die CSECT-Anweisung, die den ersten Programmabschnitt bezeichnet, einen Namen haben. Für Assembler-Programme, deren erster Programmabschnitt unbenannt ist, wird keine LSD-Information abgelegt (siehe AID, Testen von ASSEMBH-Programmen [2]).

>>>>> siehe auch DSECT- und XDSEC-Anweisung

CXD Speicherplatz für die Länge des Pseudoregistervektors reservieren

Funktion

Die CXD-Anweisung reserviert ein Wort, in das vom Binder die Gesamtlänge des Pseudoregistervektors eingetragen wird.

Format

Name	Operation	Operanden
[{name / .sym}]	CXD	

name Name
.sym Folgesymbol

Beschreibung

Der Wert des Namenseintrags ist die Adresse des Speicherbereichs, in den vom Binder die Länge des Pseudoregistervektors eingetragen wird.

Dieser Speicherbereich hat ein Längenmerkmal von 4 und muß an Wortgrenze ausgerichtet sein.

Programmierhinweis

Die CXD-Anweisung kann an beliebiger Stelle im Quellprogramm auftreten.

>>>>> siehe auch DXD-Anweisung

DC Konstante definieren

Funktion

Die DC-Anweisung definiert Konstanten im Speicher.

Format

Name	Operation	Operanden
[{name / .sym}]	DC	{[dup]typ[Ln_1][Sn_2][En_3]{'chrcon' / 'datcon[,...]' / (adrcon[,...])}[,...]

name Name
.sym Folgesymbol
dup Wiederholungsfaktor
dezimaler selbstdefinierender Wert oder positiver absoluter Ausdruck in Klammern,
Wertebereich: 0 bis 2^{24}-1
typ Konstantentyp
ein einzelner Buchstabe, kann für Zeichenkonstanten entfallen
Ln_1 Längenfaktor
n_1 ist ein dezimaler selbstdefinierender Wert oder ein positiver absoluter Ausdruck in Klammern
Sn_2 Skalenfaktor
En_3 Exponentenfaktor
n_2, bzw. n_3 ist ein positiver oder negativer selbstdefinierender Wert oder ein absoluter Ausdruck in Klammern
chrcon Wert der Zeichenkonstanten
datcon Wert der dezimalen, sedezimalen, binären, Festpunkt- und Gleitpunktkonstanten
adrcon Wert der Adreßkonstanten

Für jeden einzelnen Operanden der DC-Anweisung bzw. für jeden definierten Wert innerhalb eines Operanden wird eine eigene Konstante generiert.

Beschreibung

name ist der Name der Konstanten, bzw. der Name der ersten von mehreren Konstanten.
Der Wert des Namens ist die Adresse des höchstwertigen Bytes der ersten Konstanten.

Das Längenmerkmal des Namens ist gleich der explizit im Längenfaktor definierten Länge der Konstanten. Wenn kein Längenfaktor angegeben wurde, ist das Längenmerkmal gleich der impliziten Länge der Konstanten.

Gibt es mehr als einen Wert bzw. mehr als einen Operanden, dann ist das Längenmerkmal von name die Länge in Byte der ersten definierten Konstanten.
Man muß dann die Länge der ersten Konstanten zu name addieren, um die weiteren Konstanten anzusprechen.

dup gibt an, wie oft eine Konstante erzeugt werden soll.

Ein **Wiederholungsfaktor mit dem Wert Null** ist erlaubt und hat folgende Wirkung: Es wird kein Wert übersetzt, die Konstante wird aber entsprechend ihrem Typ ausgerichtet.

typ bestimmt den Typ der definierten Konstanten. Wenn kein Längenfaktor angegeben ist, bestimmt der Typ die Ausrichtung der Konstanten im Speicher und ihre Länge.

mögliche Konstantentypen:

C	Zeichenkonstante
B	binäre Konstante
P, Z	dezimale Konstanten
X	sedezimale Konstanten
F, H	Festpunktkonstanten
E, D, L	Gleitpunktkonstanten
A, Y, S, V, Q	Adreßkonstanten

(siehe Konstantentypen).

Ln_1, Sn_2, En_3
siehe "Modifizierfaktoren" in der DC-Anweisung

chrcon, datcon, adrcon
chrcon, datcon und adrcon sind die Werte der Konstanten.

Bei Angabe von mehreren Werten gelten die beschriebenen Merkmale für jeden einzelnen Wert.

Ausrichtung von Konstanten

Die Ausrichtung von Konstanten, also die Erhöhung des Adreßpegels auf eine bestimmte Grenze, ist abhängig vom Konstantentyp. Wenn ein Längenfaktor angegeben wurde, erfolgt in keinem Fall eine Ausrichtung.

Enthält der Operand mehr als eine Konstante, dann wird nur die erste Konstante ausgerichtet.

Typ	Ausrichtung auf
C	Byte-Grenze
X	Byte-Grenze
B	Byte-Grenze
P	Byte-Grenze
Z	Byte-Grenze
F	Wort-Grenze
H	Halbwort-Grenze
E	Wort-Grenze
D	Doppelwort-Grenze
L	Doppelwort-Grenze
A	Wort-Grenze
Y	Halbwort-Grenze
S	Halbwort-Grenze
V	Wort-Grenze
Q	Wort-Grenze

Tabelle 4-4 Ausrichtung von DC-Konstanten

Auffüllen und Abschneiden von Konstanten

Ist für eine Konstante mehr Platz vorgesehen, als für ihren Wert nötig ist, dann wird der zusätzliche Platz aufgefüllt.

Ist für eine Konstante zuwenig Platz vorgesehen, wird sie abgeschnitten und ein Teil der Konstante geht verloren.

Typ	Auffüllen	Abschneiden
C	rechts mit Leerzeichen (X'40')	rechts
X	links mit binären Nullen	links
B	links mit binären Nullen	links
P	links mit binären Nullen	links
Z	links mit EBCDIC-Nullen (X'F0')	links
F	links gemäß Vorzeichenbit	links
H	links gemäß Vorzeichenbit	links
E	rechts mit binären Nullen	nicht anwendbar } (01)
D	rechts mit binären Nullen	nicht anwendbar } (01)
L	rechts mit binären Nullen	nicht anwendbar } (01)
A	links mit binären Nullen	links
Y	links mit binären Nullen	links
S	links mit binären Nullen	links
V	links mit binären Nullen	links
Q	links mit binären Nullen	links

(01) Gleitpunktkonstanten werden nicht abgeschnitten, sondern als Fehler gemeldet und nicht übersetzt.

Tabelle 4-5 Auffüllen und Abschneiden von DC-Konstanten

Speicherplatz

Der Speicherplatz, der pro Operand einer DC-Anweisung reserviert wird, ergibt sich nach folgendem Schema:

```
Längenfaktor x Anzahl der Werte x Wiederholungsfaktor
+ alle Bytes, die zur Ausrichtung übersprungen wurden.
```

Ist mehr als ein Operand angegeben, dann ergibt sich der benötigte Speicherplatz aus der Summe des Speicherplatzbedarfs für die einzelnen Operanden.

Bezugnahme auf den Adreßpegel

Bei der Ausrichtung einer Konstanten wird der Adreßpegel auf die entsprechende Grenze erhöht.

Bezieht sich eine Adreßkonstante auf den Adreßpegel, dann wird als Wert die Speicheradresse des ersten Bytes der Konstanten benutzt. D.h., der Wert des Adreßpegels ändert sich von einer Konstanten zur nächsten um die Länge der Konstanten, wenn sich in einer DC-Anweisung mehrere Adreßkonstanten auf den Adreßpegel beziehen.

Wenn man eine Konstante, die sich auf den Adreßpegel bezieht, mit einem Wiederholungsfaktor angibt, dann wird die Konstante mit dem jeweils neuen Wert des Adreßpegels wiederholt.

Beispiele

```
Name      | Operation | Operanden
----------|-----------|-----------
DUP       | EQU       | 2
          | .         |
          | .         |
CHRCON1   | DC        | C'ABC'              (01)
CHRCON2   | DC        | 2CL5'ABC'           (02)
CHRCON3   | DC        | (DUP)CL5'ABC'       gleiche Bedeutung wie CHRCON2
          | .         |
          | .         |
HEXCON1   | DC        | X'99'               (03)
HEXCON2   | DC        | X'99,F7D5,0'        (04)
HEXCON3   | DC        | XL3'A6F4E'          (05)
          | .         |
          | .         |
ADRCON1   | DC        | A(CHRCON1)          (06)
ADRCON2   | DC        | A(*+4096)           (07)
ADRCON3   | DC        | A(*+4096,*)         (08)
```

	erzeugte Konstanten	Erklärung
(01)	C1C2C3	Länge: 3 byte
(02)	C1C2C34040C1C2C34040	2 Konstanten, beide auf Länge 5 aufgefüllt
(03)	99	Länge: 1 byte
(04)	99F7D500	3 Konstanten, Länge: 1 byte, 2 byte, 1 byte, Adresse der 2., bzw. 3. Konstanten: HEXCON2+1, bzw. HEXCON2+3
(05)	0A6F4E	Länge: 3 byte, die Konstante ist links aufgefüllt
(06)	Adresse von CHRCON1	Länge: 4 byte
(07)	Adresse des ersten Bytes von ADRCON2 + 4096	Länge: 4 byte
(08)	Wert der ersten Konstanten:	Adresse des ersten Bytes von ADRCON3 + 4096,
	Wert der zweiten Konstanten:	Adresse des ersten Bytes von ADRCON3 + 4, beide Konstanten haben eine Länge von 4 byte.

Modifizierfaktoren

Die Modifizierfaktoren einer Konstanten sind die Länge in byte, die eine Konstante erhalten soll, der Skalenfaktor und der Exponentenfaktor.

Längenfaktor

Der Längenfaktor überschreibt die implizite Länge einer Konstanten. Er gibt die Anzahl von Bytes an, die für eine Konstante reserviert werden. Er bestimmt daher, ob eine Konstante aufgefüllt oder ihr Wert abgeschnitten wird. Bei Angabe eines Längenfaktors wird die Konstante nicht ausgerichtet.

`Format des Längenfaktors` Ln_1

n_1 ist ein dezimaler selbstdefinierender Wert oder ein positiver absoluter Ausdruck in Klammern.

n_1 darf den Maximalwert nicht überschreiten, der für die verschiedenen Konstantentypen zulässig ist.

Typ	implizite Länge	mögliche Werte für n_1 (byte)
C	} (01)	1 bis 256
X		1 bis 256
B		1 bis 256
P		1 bis 16
Z		1 bis 16
F	4	1 bis 8
H	2	1 bis 8
E	4	1 bis 8
D	8	1 bis 8
L	16	1 bis 16
A	4	1 bis 4
Y	2	1 oder 2
S	2	2
V	4	3 oder 4
Q	4	1 bis 4

(01) Die implizite Länge wird, je nach Länge der definierten Konstanten, berechnet

Tabelle 4-6 Längenfaktoren bei DC-Konstanten

Skalenfaktor

Der Skalenfaktor kann nur bei Festpunkt- und Gleitpunktkonstanten verwendet werden. Er definiert die interne Stellenverschiebung für eine Konstante, und zwar die Anzahl von Binärstellen bei Festpunktkonstanten und die Anzahl von Sedezimalstellen bei Gleitpunktkonstanten.

Format des Skalenfaktors Sn_2

n_2 ist ein positiver oder negativer selbstdefinierender Wert oder ein absoluter Ausdruck in Klammern.

n_2 kann ein Vorzeichen enthalten. Wenn kein Vorzeichen angegeben ist, dann wird ein Pluszeichen angenommen.

Typ	mögliche Werte für n_2
F	-187 bis +346
H	-187 bis +346
E	0 bis +14
D	0 bis +14
L	0 bis +28

Tabelle 4-7 Höchstwerte der Skalenfaktoren bei Festpunkt- und Gleitpunktkonstanten

Skalenfaktor in Festpunktkonstanten

Der Skalenfaktor gibt hier die Zweierpotenz an, mit der eine Konstante multipliziert werden soll, nachdem ihr Wert in die interne binäre Darstellung umgewandelt wurde, aber bevor er in stellengerechte Position übersetzt wurde.

Die Multiplikation einer Binärzahl mit einer Zweierpotenz bewirkt die Verschiebung des Binärpunktes weg von seiner ursprünglich angenommenen Lage rechts hinter der letzten Stelle.

Deshalb gibt der Skalenfaktor folgendes an:

Wenn n_2 positiv ist	die Anzahl von binären Stellen, die der Bruchteil der Binärzahl belegen soll. Hier wird der ganzzahlige Teil der Konstanten also nach links verschoben.
Wenn n_2 negativ ist	die Anzahl der binären Stellen, die vom ganzzahligen Teil der Binärzahl gestrichen werden sollen. Hier wird der ganzzahlige Teil der Konstanten also nach rechts verschoben.

Falls wegen des Skalenfaktors oder wegen seines Fehlens Stellen verloren gehen, wird die am weitesten rechts stehende Stelle der Binärzahl gerundet (bei einer Zahl > 5 wird aufgerundet, bei einer Zahl < 5 wird abgerundet).

Wird bei einer Festpunktkonstanten, die Kommastellen enthält, kein Skalenfaktor angegeben, dann gehen die Stellen hinter dem Komma verloren.

Skalenfaktor in Gleitpunktkonstanten

n_2 gibt hier die Anzahl der Sedezimalstellen an, um die die Mantisse in der binären Darstellung einer Gleitpunktkonstanten nach rechts verschoben werden soll. Die erste Stelle der Mantisse wird ursprünglich direkt hinter dem sedezimalen Punkt angenommen (normalisierte Gleitpunktkonstante).

Durch den Skalenfaktor wird eine nicht normalisierte Gleitpunktkonstante erzeugt, d.h., daß die am weitesten links stehenden Stellen der Mantisse sedezimale Nullen enthalten.

n_2 muß hier positiv sein.

Wenn durch den Skalenfaktor die Mantisse verschoben wird, dann wird die Charakteristik der Gleitpunktkonstanten entsprechend korrigiert.

Gehen auf Grund der Angabe eines Skalenfaktors Stellen verloren, dann wird entsprechend der am weitesten links stehenden Stelle des verlorenen Teils gerundet.

Exponentenfaktor

Der Exponentenfaktor kann nur bei Festpunkt- und Gleitpunktkonstanten verwendet werden. Er definiert die Zehnerpotenz, mit der der Wert einer Konstanten multipliziert werden soll, bevor er in seine interne binäre Darstellung umgewandelt wird.

`Format des Exponentenfaktors` En_3

n_3 ist ein positiver oder negativer selbstdefinierender Wert oder ein absoluter Ausdruck in Klammern

n_3 kann ein Vorzeichen enthalten. Wenn kein Vorzeichen angegeben ist, wird ein Pluszeichen angenommen.

Der mögliche Bereich für n_3 ist -85 bis +75.

Hinweis

En_3 darf nicht mit dem Exponenten verwechselt werden, der im Feld datcon für einen Wert definiert werden kann. Gibt es in einem Operanden beide Arten von Exponentendefinitionen, werden ihre Werte zusammengerechnet, bevor der Wert in das binäre Format umgewandelt wird. Diese Exponentensumme muß auch noch im erlaubten Bereich von -85 bis +75 liegen.

Konstantentypen

Die Angabe typ in der DC-Anweisung bestimmt den Typ der definierten Konstanten (siehe Tab.4-7). Davon ausgehend kann der Assembler die Konstante interpretieren und in das entsprechende Maschinenformat übersetzen. Wenn kein Längenfaktor angegeben ist, bestimmt der Typ die Ausrichtung der Konstanten im Speicher und den Speicherplatz, den die Konstante belegt.

Code	Konstantentyp	Maschinenformat
C	Zeichenkonstante	8-bit-Code für jedes Zeichen
X	Sedezimale Konstante	4-bit-Code für jedes Sedezimalzeichen
B	Binäre Konstante	Binärformat
F	Festpunktkonstante	Binäres Festpunktformat mit Vorzeichen, ein Wort
H	Festpunktkonstante	Binäres Festpunktformat mit Vorzeichen, ein Halbwort
E	Gleitpunktkonstante	Kurzes Gleitpunktformat, ein Wort
D	Gleitpunktkonstante	Langes Gleitpunktformat, ein Doppelwort
L	Gleitpunktkonstante	erweitertes Gleitpunktformat, zwei Doppelworte
P	Dezimale Konstante	Gepacktes Dezimalformat
Z	Dezimale Konstante	Ungepacktes Dezimalformat
A	Adreßkonstante	Adreßwert, ein Wort
Y	Adreßkonstante	Adreßwert, ein Halbwort
S	Adreßkonstante	Basisregister und Distanzwert, ein Halbwort
V	Adreßkonstante	Reservierter Speicherplatz für externe symbolische Adressen; jede Adresse ein Wort
Q	Adreßkonstante	Reservierter Speicherplatz für den Offset eines Pseudoregisters bezüglich des Anfangs des Pseudoregistervektor

Tabelle 4-8 Konstantentypen

Im folgenden werden die Konstantentypen einzeln beschrieben.

Zeichenkonstanten C

Mit der Zeichenkonstante kann man Zeichenfolgen definieren. Zeichenkonstanten können mit jedem abdruckbaren Zeichen gebildet werden. Jedes Zeichen, das im Feld chrcon angegeben wurde, wird in ein Byte übersetzt.

Es erfolgt keine Ausrichtung im Speicher.

Der größte erlaubte Längenfaktor ist 256.
Ist kein Längenfaktor angegeben, dann entspricht die Länge der Konstanten der Anzahl der Zeichen in chrcon.
Ist ein Längenfaktor angegeben, dann wird der Wert der Konstanten

- rechts abgeschnitten, wenn der Längenfaktor zu klein ist und
- rechts mit Leerzeichen (X'40') aufgefüllt, wenn der Längenfaktor größer als die Anzahl der Zeichen ist.

Für die Darstellung von Hochkomma (') und kommerziellem Und (&) gilt:
Das Hochkomma (') wird in der Assemblersprache und das kommerzielle Und (&) in der Makrosprache als syntaktisches Zeichen verwendet. Deshalb müssen für jedes Hochkomma oder jedes &-Zeichen, die in einer Zeichenkonstanten verwendet werden sollen, zwei Hochkommas bzw. zwei &-Zeichen geschrieben werden. Die beiden Hochkommas bzw. &-Zeichen werden als einzelnes Hochkomma, bzw. als einzelnes & übersetzt.

Beispiele

```
Name     | Operation | Operanden
---------|-----------|-------------------
CHRCON1  | DC        | C'ABC,DEF'          (01)
CHRCON2  | DC        | C'&& ABC,DEF'       (02)
CHRCON3  | DC        | 3CL4'12345'         (03)
```

	erzeugte Konstanten	Erklärung
(01)	C1C2C36BC4C5C6	Länge: 7 byte, das Komma wird als Zeichen interpretiert.
(02)	5040C1C2C36BC4C5C6	Länge: 9 byte, die zwei &-Zeichen gelten als ein Zeichen.
(03)	F1F2F3F4F1F2F3F4F1F2F3F4	Konstante mit Wiederholungs- und Längenfaktor; die Länge ist zu kurz definiert, die Konstante wird rechts abgeschnitten.

Sedezimalkonstanten X

Eine sedezimale Konstante besteht aus einer oder mehreren Sedezimalziffern. Jede Sedezimalstelle, die in datcon definiert ist, wird in vier Bits übersetzt. Bei einer ungeraden Anzahl von Stellen werden die linken vier Bits des höchstwertigen Bytes mit einer sedezimalen Null aufgefüllt.

Der größte Längenfaktor ist 256 (byte).
Ist kein Längenfaktor angegeben, dann ist die implizite Länge der Konstanten gleich der Anzahl der Bytes in datcon. Ist ein Längenfaktor angegeben, dann wird die Konstante

- links abgeschnitten, wenn der Längenfaktor zu klein ist und
- links mit sedezimalen Nullen aufgefüllt, wenn der Längenfaktor größer ist, als die Hälfte der Anzahl der sedezimalen Stellen.

Beispiele

```
Name        | Operation | Operanden
------------+-----------+-----------------
HEXCON1     | DC        | X'FF00FFFF'        (01)
HEXCON2     | DC        | 3XL2'BD8E7'        (02)
HEXCON3     | DC        | 3X'BD8E7'          (03)
```

	erzeugte Konstanten	Erklärung
(01)	FF00FFFF	Die Konstante erzeugt das Bitmuster eines Wortes. HEXCON1 setzt das erste, dritte und vierte Byte eines Wortes auf 1.
(02)	D8E7D8E7D8E7	Konstante mit Wiederholungs- und Längenfaktor, die Länge ist zu kurz definiert, daher wird links abgeschnitten
(03)	0BD8E70BD8E70BD8E7	Die Länge wird implizit berechnet, links wird mit einer sedezimalen Null aufgefüllt

Binärkonstanten B

Mit einer Binärkonstante kann man beliebige Bitmuster definieren. Sie besteht aus einer Folge der Binärziffern 0 und 1.

Der größtmögliche Längenfaktor ist 256 byte.
Ist kein Längenfaktor angegeben, dann ist die implizite Länge der Konstanten die Anzahl von Bytes, die von der Konstanten belegt werden. Dabei wird jeweils links mit binären Nullen aufgefüllt, wenn der Wert der Konstanten keine ganzzahlige Anzahl von Bytes belegt. Ist ein Längenfaktor angegeben, dann wird der Wert der Konstanten

- links abgeschnitten, wenn der Längenfaktor zu klein ist und
- links mit binären Nullen aufgefüllt, wenn der Längenfaktor grösser angegeben ist, als für die Aufnahme der definierten Bits notwendig ist.

Beispiele

```
Name      | Operation | Operanden
----------|-----------|-------------------
BCON      | DC        | B'10101101'          (01)
BSHORT    | DC        | BL1'111100011'       (02)
BLONG1    | DC        | BL1'110'             (03)
BLONG2    | DC        | B'110'               (04)
```

	erzeugte Konstanten	Erklärung
(01)	AD	Länge: 1 byte
(02)	E3	Die Konstante wird links abgeschnitten
(03)	06	Die Konstante wird links mit binären Nullen aufgefüllt
(04)	06	Gleiche Wirkung wie BLONG1, die Länge wird implizit berechnet

Festpunktkonstanten F und H

Festpunktkonstanten definieren Daten, die in Festpunktbefehlen verwendet werden können.

Der Wert einer Festpunktkonstanten wird als Dezimalzahl geschrieben, der ein Dezimalexponent folgen kann:

- Die Dezimalzahl kann ganzzahlig, gebrochen oder gemischt sein und ein positives oder negatives Vorzeichen haben. Fehlt das Vorzeichen, wird sie als positiv interpretiert. Fehlt der Dezimalpunkt, dann wird die Dezimalzahl als ganzzahlig interpretiert.
- Wird für den Wert der Konstanten ein Dezimalexponent angegeben, dann muß er direkt auf die Zahl folgen. Er wird als En geschrieben, wobei n eine Dezimalzahl sein muß, die ein Vorzeichen haben kann. En wird als Exponent zur Basis 10 aufgefasst. Die Summe aus Exponent und Exponentenfaktor darf den zulässigen Bereich von -85 bis +75 nicht überschreiten.

Übersetzung von Festpunktkonstanten

1. Die definierte Zahl, multipliziert mit 10 hoch dem Exponenten, wird in eine binäre Zahl mit Vorzeichen umgewandelt. Das Vorzeichen wird im höchstwertigen Bit verschlüsselt.
   Eine negative Zahl wird als Zweierkomplement mit einem auf 1 gesetzten Vorzeichenbit dargestellt.
2. Diese Zahl wird entsprechend dem angegebenen Skalenfaktor ausgerichtet. Wird eine gebrochene oder eine gemischte Zahl ohne Skalenfaktor angegeben, dann geht der gebrochene Teil verloren.
3. Der binäre Wert wird, wenn nötig, gerundet.
4. Nachdem die Konstante übersetzt ist, wird der Wiederholungsfaktor ausgewertet.

Ausrichtung, Länge, Wertebereich

Die Konstante wird auf Wort- bzw. Halbwortgrenze ausgerichtet, wenn kein Längenfaktor angegeben ist.

Die implizite Länge für Wortkonstanten (F) beträgt vier byte, für Halbwortkonstanten (H) zwei byte. Mit einem Längenfaktor können jedoch für beide Konstantentypen Längen bis zu acht byte angegeben werden. Dadurch ergibt sich für Festpunktkonstanten folgender darstellbarer Wertebereich:

Länge	darstellbarer Wertebereich
8	-2^{63} bis $2^{63}-1$
4	-2^{31} bis $2^{31}-1$
2	-2^{15} bis $2^{15}-1$
1	-2^{7} bis $2^{7}-1$

Tabelle 4-9 Darstellbarer Wertebereich für Festpunktkonstanten

Der Wertebereich hängt ab von der implizit oder explizit definierten Länge.
Ist die Konstante kleiner als ihre definierte Länge, wird sie links gemäß Vorzeichenbit aufgefüllt.
Überschreitet die Konstante den Wertebereich, der auf Grund ihrer definierten Länge möglich ist, dann geht das Vorzeichen verloren, die Konstante wird links abgeschnitten.

Beispiele

```
Name     | Operation | Operanden
---------+-----------+-----------------
HCON1    | DC        | H'+12'          (01)
HCON2    | DC        | H'-12'          (02)
FCON1    | DC        | F'12.3'         (03)
FCON2    | DC        | FS8'12.3'       (04)
FCON3    | DC        | FS8'123'        (05)
```

	erzeugte Konstanten	Erklärung
(01)	000C	Länge: 2 byte
(02)	FFF4	Der negative Wert wird als Zweierkomplement dargestellt, das Vorzeichenbit steht auf 1.
(03)	0000000C	Wortkonstante; weil kein Skalenfaktor angegeben ist, geht der Bruchteil verloren; es wird abgerundet.
(04)	00000C4D	Der Skalenfaktor reserviert 1 byte für den Bruchteil, der ganzzahlige Teil der Konstante wird nach links verschoben; es wird aufgerundet.
(05)	00007B00	Der Skalenfaktor reserviert 1 byte für den Bruchteil, auch wenn der Wert keinen enthält.

Gleitpunktkonstanten E, D und L

Gleitpunktkonstanten definieren Daten, die in Gleitpunktbefehlen verwendet werden können.

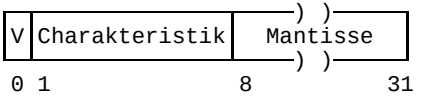
Der Wert einer Gleitpunktkonstanten wird als Dezimalzahl geschrieben, der ein Exponent folgen kann:

- Die Dezimalzahl kann ganzzahlig, gebrochen oder gemischt sein und ein positives oder negatives Vorzeichen haben. Fehlt das Vorzeichen, wird die Zahl als positiv aufgefaßt. Fehlt der Dezimalpunkt, wird die Zahl als ganzzahlig interpretiert.
- Wird für die Zahl ein Exponent angegeben, dann muß er direkt folgen und in der Form En geschrieben sein. n muß eine Dezimalzahl sein, die ein Vorzeichen haben kann. En wird als Exponent zur Basis 10 aufgefasst.
  Die Summe aus Exponent und Exponentenfaktor darf den zulässigen Bereich von -85 bis +75 nicht überschreiten.

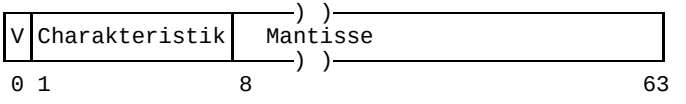
Maschinenformat von Gleitpunktkonstanten

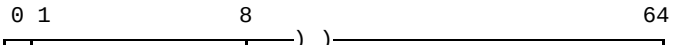
Das Maschinenformat für eine Gleitpunktkonstante besteht aus zwei Teilen: dem Exponententeil (Charakteristik), gefolgt von dem gebrochenen Teil (Mantisse). Ein Vorzeichen-Bit zeigt an, ob eine positive oder negative Zahl definiert wurde (siehe Assemblerbefehle, Beschreibung [3]).

kurzes Gleitpunktformat: (4 byte)

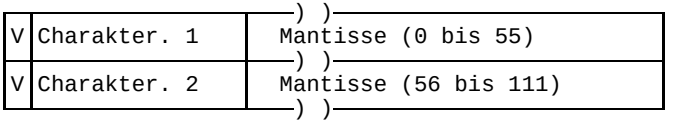


langes Gleitpunktformat: (8 byte)





erweitertes Gleitpunktf.: (16 byte)



- Das Vorzeichen im zweiten Doppelwort ist gleich dem im ersten
- Charakteristik 2 = (Charakteristik 1 - 14)

Übersetzung von Gleitpunktkonstanten

Eine Gleitpunktkonstante muß vor der Umwandlung in das Maschinenformat erst in einen Bruch konvertiert werden. Der Exponent wird dann in sein Binäräquivalent umgesetzt und der Bruch in eine Binärzahl konvertiert.

Der Assembler übersetzt eine Gleitpunktkonstante in ihre binäre Darstellung wie es an folgendem *Beispiel* erklärt wird:

Die Gleitpunktkonstante KON wurde folgendermaßen definiert:

Name	Operation	Operanden
KON	DC	E'-167.0'

1. $-167.0_{10} = -A7_{16}$ Umwandlung des Bruchs in eine Binärzahl
2. $-A7_{16} = (-0.A7*16^2)_{16}$ Normalisierung
3. Charakteristik = (Exponent + 64 = $66_{)1}0 = 4_{21}6$
4. Wegen des negativen Vorzeichen wird das höchstwertige Bit auf 1 gesetzt, das entspricht einer Addition von 80_{16}:

   $80_{16} + 42_{16} = C2_{16}$
5. Eine E-Konstante hat eine implizite Länge von 4 byte, d.h. das erste Byte enthält Charakteristik und Vorzeichen, die rechten 3 Bytes enthalten die Mantisse:

   C2 A7 00 00 d.h. C2 = Vorzeichen + Charakteristik
   A7 00 00 = Mantisse
6. Wurde ein Skalenfaktor definiert, dann werden links an den normalisierten gebrochenen Teil sedezimale Nullen angehängt und die Charakteristik entsprechend angeglichen.
7. Der gebrochene Teil wird entsprechend der impliziten, bzw. expliziten Länge der Konstanten gerundet.
8. Ein negativer gebrochener Teil wird nicht als Zweierkomplement dargestellt, sondern in direkter Form mit Vorzeichen.
9. Nachdem die Konstante übersetzt ist, wird der Wiederholungsfaktor ausgewertet.

Ausrichtung, Länge, Wertebereich

E-Konstanten werden an Wort-Grenzen ausgerichtet, D- und L-Konstanten an Doppelwortgrenzen. Wenn ein Längenfaktor definiert ist, erfolgt keine Ausrichtung.

Die implizite Länge beträgt für E-Konstanten 4 byte, für D-Konstanten 8 byte und für L-Konstanten 16 byte. Mit einem Längenfaktor kann für E- und D-Konstanten jede Länge bis zu 8 byte angegeben werden, für L-Konstanten bis zu 16 byte. Dadurch ergibt sich für die Mantisse von Gleitpunktkonstanten ein Wertebereich, der übersetzt werden kann:

Typ	Wertebereich der Mantisse (exakt)	Wertebereich (annähernd)
E	16^{-65} bis $(1-16^{-6})*16^{63}$	$5{,}4*10^{-75}$ bis $7{,}2*10^{75}$
D	16^{-65} bis $(1-16^{-14})*16^{63}$	
L	16^{-65} bis $(1-16^{-28})*16^{63}$	

Tabelle 4-10 Darstellbarer Wertebereich der Mantisse in Gleitpunktkonstanten

Liegt der Wert, der für eine Konstante definiert wurde, nicht innerhalb dieser Grenzen, dann wird die Konstante nicht übersetzt und mit einer Fehlermeldung versehen.

Beispiele

```
Name     | Operation | Operanden
---------+-----------+-----------------
ECON1    | DC        | E'167'              (01)
ECON2    | DC        | E'16.7'             (02)
ECON3    | DC        | ES2'16.7'           (03)
ECON4    | DC        | EE10'16.7'          (04)
ECON5    | DC        | E'16.7E10'          (05)
```

	erzeugte Konstanten	Erklärung
(01)	42A70000	Gleitpunktkonstante der Länge 4 byte, das erste Byte enthält Charakteristik und Vorzeichen, die rechten 3 Bytes enthalten die Mantisse.
(02)	4210B33	Die Charakteristik bleibt gleich, durch die Angabe von Kommastellen ändert sich die Mantisse.
(03)	440010B33	Der Skalenfaktor bewirkt eine Verschiebung des gebrochenen Teils um zwei Sedezimalstellen nach rechts. Die Charakteristik wird geändert weil sich durch die Verschiebung der Wert des Exponenten ändert.
(04)	4A26E1FA	Wegen des Exponentenfaktors wird der Wert 16.7 mit 10^{10} multipliziert, bevor er umgewandelt wird.
(05)	4A26E1FA	hat das gleiche Ergebnis wie ECON4. Übersetzt wird $16.7*10^{10}$.

Dezimalkonstanten P und Z

Die Dezimalkonstanten definieren Daten, die in Dezimalbefehlen verwendet werden können.

Der Wert einer Dezimalkonstanten wird als Dezimalzahl geschrieben, die ein Vorzeichen haben kann. Ist kein Vorzeichen angegeben, wird die Zahl als positiv interpretiert. Die Dezimalzahl kann einen Dezimalpunkt enthalten. Er wird jedoch bei der Übersetzung der Konstanten in das interne Format nicht berücksichtigt.

Der größte Längenfaktor ist 16 byte.
Ist kein Längenfaktor angegeben, dann entspricht die implizite Länge der Konstanten der Anzahl von Bytes, die sie belegt.
Ist ein Längenfaktor angegeben, dann wird der Wert der Konstanten

- links abgeschnitten, wenn die Konstante mehr Bytes erfordert, als der Längenfaktor angibt und
- links aufgefüllt, wenn der Längenfaktor grösser ist als nötig. Bei Z-Konstanten wird mit einer dezimalen Null aufgefüllt, bei P-Konstanten werden die Bits jedes angefügten Bytes auf Null gesetzt.

Übersetzung von Dezimalkonstanten

Bei gepackten Dezimalkonstanten (P) wird jede Ziffer in ihr 4 bit langes binäres Äquivalent übersetzt. Das Vorzeichen kommt in die am weitesten rechts stehenden 4 Bits der Konstanten.
Wird eine gerade Anzahl von gepackten Dezimalstellen definiert, werden die linken 4 Bits des am weitesten links stehenden Bytes auf Null gesetzt, und die rechten 4 Bits enthalten die erste Stelle der Dezimalzahl.

Bei ungepackten Dezimalkonstanten (Z) wird jede Ziffer in ihre 8 bit lange EBCDIC-Darstellung umgewandelt. Das Vorzeichen steht in den ersten 4 Bits des am weitesten rechts stehenden Bytes der Konstanten.

Bei beiden Konstantentypen wird ein Plus als C, ein Minus als D abgelegt.

Beispiele

```
Name     | Operation | Operanden
---------+-----------+-----------------
PCON1    | DC        | P'+153'          (01)
PCON2    | DC        | P'-153'          (02)
ZCON1    | DC        | Z'-153'          (03)
ZCON2    | DC        | Z'-1.53'         (04)
```

	erzeugte Konstanten	Erklärung
(01)	153C	Länge: 2 byte
(02)	153D	Länge: 2 byte
(03)	F1F5D3	Länge: 3 byte
(04)	F1F5D3	Länge: 3 byte, der Dezimalpunkt wird ignoriert.

Adreßkonstanten

Adreßkonstanten enthalten Speicheradressen. Man kann Adreßkonstanten verwenden, um z.B. Basisregister zu initialisieren oder zur Verknüpfung von Programmabschnitten.

Adreßkonstanten vom Typ A und Y

Der Wert einer A- oder Y-Konstanten, der in adrcon definiert wird, kann ein absoluter oder ein relativer Ausdruck sein.

A-Konstanten werden auf Wortgrenze und Y-Konstanten auf Halbwortgrenze ausgerichtet, wenn kein Längenfaktor angegeben ist.

Die implizite Länge von A-Konstanten beträgt 4 byte, von Y-Konstanten 2 byte. Mit dem Längenfaktor können für A-Konstanten Längen von 1 bis 4 byte angegeben werden, für Y-Konstanten 1 bis 2 byte.

Übersetzung von A- und Y-Konstanten

– adrcon ist ein absoluter Ausdruck:

  Der Wert in adrcon wird auf 32 bit berechnet und dann links abgeschnitten oder links aufgefüllt, wenn die Länge der Konstanten es erfordert.

– adrcon ist ein relativer Ausdruck:

  Der Assembler verwendet den Adreßpegel als vorläufigen Wert. Der endgültige Wert der Konstanten wird erst beim Laden des Programms in die Konstante eingesetzt.

Programmierhinweise

1. Für 31-bit-Adressierung dürfen nur 4 byte lange A-Konstanten verwendet werden.
2. Ein relativer Ausdruck in einer A- oder Y-Konstanten kann externe Namen enthalten und dadurch eine Adresse in einem unabhängig übersetzten Programm bezeichnen. In diese Adreßkonstante trägt der Assembler den vorläufigen Wert Null ein (siehe Beispiel 03).
3. Y-Konstanten werden häufig verwendet bei Offset-Adressierung innerhalb von Tabellen oder innerhalb von Datenbereichen, die über den Ablaufteil-Makro REQM bereitgestellt wurden (siehe Makroaufrufe an den Ablaufteil, Beschreibung [6]).

Beispiele

```
Name      | Operation | Operanden
----------+-----------+-----------------------
ACON1     | DC        | A(*+4096)            (01)
ACON2     | DC        | A(ACON1)             (02)
          | .         |                       ⎫
          | .         |                       ⎪
PROGRA    | START     |                       ⎪
          | .         |                       ⎪
          | EXTRN     | MARK                  ⎪
          | .         |                       ⎪
          | L         | 15,ACON               ⎪
          | .         |                       ⎪
ACON      | DC        | A(MARK)               ⎪
          | .         |                       ⎪
          | END       |                       ⎬ (03)
          | .         |                       ⎪
          | .         |                       ⎪
PROGRB    | START     |                       ⎪
          | .         |                       ⎪
          | ENTRY     | MARK                  ⎪
          | .         |                       ⎪
MARK      | EQU       | *                     ⎪
          | .         |                       ⎪
          | END       |                       ⎭
```

	Wert der Konstanten	Erklärung
(01)	Adreßpegel von ACON1 + 4096	Der relative Ausdruck in der Konstanten enthält einen Verweis auf den Adreßpegel. Der Wert von * ist der Adreßpegel von ACON1.
(02)	Adreßpegel von ACON1	
(03)	Null	Mögliche Verknüpfung von zwei Programmen. Die A-Konstante in PROGRA ist als EXTRN-Adresse (Sprungdresse) gekennzeichnet, in PROGRB als ENTRY-Adresse (Einsprungadresse). Das Sprungziel liegt in PROGRB.

Adreßkonstanten vom Typ S

S-Konstanten ermöglichen es, Adressen in Basis-Distanz-Form abzulegen.

Der Wert einer S-Konstanten kann als symbolische oder als nicht-symbolische Adresse angegeben werden. Eine symbolische Adresse wird vom Assembler in Basisregister und Distanzwert zerlegt, eine nicht-symbolische Adresse muß in der Form (Distanz(Basisregister)) angegeben werden.

Eine S-Konstante wird auf Halbwortgrenze ausgerichtet, ihre implizite Länge ist 2 byte. Die am weitesten links stehenden 4 Bits der übersetzten Konstanten enthalten die Nummer des Basisregisters und die restlichen 12 Bits den Distanzwert.

S-Konstanten dürfen nicht in Literalen verwendet werden.

Programmierhinweis

S-Konstanten werden in erster Linie verwendet, wenn ein Maschinen-Befehlscode nicht über einen mnemotechnischen Operationscode erzeugt werden soll, sondern über DC-Konstanten.

Beispiel

```
Name      | Operation | Operanden
----------+-----------+-----------------------------
          |     .     |
          |     .     |
R0        | EQU       | 0
R1        | EQU       | 1
R11       | EQU       | 11
          |     .     |
          |     .     |
          | BALR      | R11,R0
          | USING     | *,R11
          |     .     |
          |     .     |
          | L         | R1,ADR                (01)
          |     .     |
          |     .     |
          | DC        | X'58'                 ┐
          | DC        | X'10'                 } (02)
          | DC        | S(ADR)                ┘
          | TERM      |
          |     .     |
          |     .     |
ADR       | DC        | C'ABCD'
          | END       |
```

	erzeugter Objektcode	Erklärung
(01)	58 10 B026	
(02)	58	Die Folge von DC-Konstanten
	10	erzeugt einen Objektcode, der
	B026	dem des L-Befehls entspricht.

Adreßkonstanten vom Typ V

Mit V-Konstanten kann man Speicherplatz für Einsprungadressen in ein anderes Programm reservieren. Man darf V-Konstanten nur als Sprungadressen verwenden, nicht zur Adressierung externer Daten.

Der Wert der Konstanten wird als Name angegeben. Dieser Name darf nicht durch eine EXTRN-Anweisung gekennzeichnet sein, da er vom Assembler automatisch als externe Adresse interpretiert wird (falls doch, wird jedoch aus Kompatibilitätsgründen kein Flag ausgegeben). Es ist zu beachten, daß die Angabe eines Namens in einer V-Konstanten nicht gleichzeitig die Definition des Namens als externes Symbol für dieses Programm bedeutet.

Der Wert einer V-Konstanten ist Null, bis das Programm geladen ist. Der korrekte Wert der Adresse wird durch den Lader eingefügt.

V-Konstanten werden auf Wortgrenze ausgerichtet, wenn kein Längenfaktor angegeben ist. Die implizite Länge einer V-Konstanten ist 4 byte. Mit dem Längenfaktor kann eine Länge von 3 oder 4 byte angegeben werden. In diesem Fall kann der relative Wert der Konstanten abgeschnitten werden.

Programmierhinweis

Für 31-bit-Adressierung dürfen nur V-Konstanten mit der Länge 4 byte verwendet werden.

Beispiel

```
Name      | Operation | Operanden
----------+-----------+-----------
PROGRA    | START     |
          | .         |
          | L         | 15,VCON
          | BALR      | 14,15
          | .         |
VCON      | DC        | V(MARK)
          | .         |
          | END       |
          | .         |                    (01)
          | .         |
PROGRB    | START     |
          | ENTRY     | MARK
          | .         |
          | USING     | MARK,15
MARK      | LA        | ...
          | .         |
          | .         |
          | BR        | 14
          | .         |
          | END       |
```

(01) Dieses Beispiel zeigt eine Programmverknüpfung, die über eine V-Konstante realisiert ist. Der Name MARK darf im gleichen Programm nicht durch eine EXTRN-Anweisung gekennzeichnet werden.

Adreßkonstanten vom Typ Q

Mit Q-Konstanten kann Speicherplatz reserviert werden, in dem der Offset eines Pseudoregisters innerhalb des Pseudoregistervektors abgelegt werden soll. Im Pseudoregistervektor liegen die Pseudoregister in einer vom Binder bestimmten Reihenfolge hintereinander. Die Q-Konstante enthält also den Abstand des Pseudoregisters zum Anfang des Pseudoregistervektors.

Der Wert, der in einer Q-Konstanten definiert wird, ist der Name eines über DXD definierten Pseudoregisters oder einer DSECT, die durch die Referenzierung in einer Q-Konstanten zusätzlich als Pseudoregister eingetragen wird. Diese Adresse wird erst vom Binder eingetragen.

Beschreibung der Verwendung von Q-Konstanten siehe 3.3.2, Pseudoregister.

Hinweis

Literale dürfen keine Q-Konstanten enthalten.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------------
DNAME     | DSECT     |
D1        | DS        | 3F
D2        | DS        | CL15
          | .         |
          | .         |
CNAME     | CSECT     |
          | .         |
          | .         |
QCON      | DC        | Q(DNAME)          (01)
          | END       |
```

(01) DNAME definiert eine DSECT. QCON enthält die Distanz von DNAME zum Anfang des Pseudoregistervektors.

DS Speicherplatz reservieren

Funktion

Die DS-Anweisung reserviert Speicherbereiche und ordnet ihnen Namen zu.

Format

Name	Operation	Operanden
[{name / .sym}]	DS	[dup]typ[Ln][{'chrcon' / 'datcon'}]

name Name
.sym Folgesymbol
dup Wiederholungsfaktor
dezimaler selbstdefinierender Wert oder positiver absoluter Ausdruck in Klammern,
Wertebereich: 0 bis 2^{24}-1
typ Typ des reservierten Speicherbereichs, ein einzelner Buchstabe
Ln Längenfaktor
n ist ein dezimaler selbstdefinierender Wert oder ein positiver absoluter Ausdruck in Klammern
chrcon Wertangabe für einen Zeichenbereich
datcon Wertangabe für dezimale, sedezimale, binäre, Festpunkt- und Gleitpunktbereiche

Die Werte, die in den Operanden der DS-Anweisung angegeben werden, werden nicht in Objektcode übersetzt. Sie werden ggf. nur zur Bestimmung impliziter Längenfaktoren herangezogen.

Hinweis

Es ist möglich, im Operanden der DS-Anweisung das ganze Format der DC-Anweisung anzugeben. Diese zusätzlichen Angaben haben aber keine Auswirkung auf den reservierten Bereich.

Beschreibung

name ist der Name des reservierten Speicherbereichs. Der Wert des Namens ist die Adresse des am weitesten links stehenden Bytes des reservierten Bereichs.

Das Längenmerkmal des Namens ist gleich der explizit im Längenfaktor definierten Länge des Bereichs. Wenn kein Längenfaktor angegeben wurde, ist das Längenmerkmal gleich der impliziten Länge des Bereichs, die abhängig ist vom angegebenen Typ.

dup gibt an, wie oft ein Bereich reserviert werden soll. Wenn für dup ein Ausdruck angegeben wird, müssen die in diesem Ausdruck verwendeten Namen nicht vorher definiert sein.

Ein **Wiederholungsfaktor mit dem Wert Null** ist erlaubt und hat folgende Wirkung: Es wird kein Speicherplatz reserviert, der Bereich wird aber entsprechend seinem Typ ausgerichtet und erhält ein Längenmerkmal.

typ Wenn kein Längenfaktor angegeben ist, bestimmt der Typ die Ausrichtung des Bereichs im Speicher und seine Länge.

mögliche Typangaben:	C	Zeichenbereiche
	B	binäre Bereiche
	P, Z	dezimale Bereiche
	X	sedezimale Bereiche
	F, H	Festpunktbereiche
	E, D, L	Gleitpunktbereiche
	A, Y, S, V, Q	Adreßbereiche

Ln gibt die Länge des Bereichs in byte an, der reserviert werden soll und überschreibt die implizite Länge.
Wenn für n ein Ausdruck angegeben wird, müssen die verwendeten Namen nicht vorher definiert sein.

Bei Angabe eines Längenfaktors wird der entsprechende Bereich nicht ausgerichtet.

Typ	implizite Länge	mögliche Werte für n (byte)
C	1	1 bis 2^{24}-1
X	1	1 bis 2^{24}-1
B	1	1 bis 256
P	1	1 bis 16
Z	1	1 bis 16
F	4	1 bis 8
H	2	1 bis 8
E	4	1 bis 8
D	8	1 bis 8
L	16	1 bis 16
A	4	1 bis 4
Y	2	1 oder 2
S	2	2
V	4	3 oder 4
Q	4	1 bis 4

Tabelle 4-11 Längenfaktoren in der DS-Anweisung

chrcon,datcon

chrcon, und datcon sind die Werte, die in den Operanden angegeben werden können.

Wenn bei den Typen C, B, P, Z und X kein Längenfaktor angegeben ist, berechnet der Assembler an Hand des angegeben Wertes die Länge des zu reservierenden Bereichs.

Wenn für einen Bereich ein Wert angegeben wird, dann muß er für den Typ des entsprechenden Bereichs zulässig sein (siehe Beschreibung der Konstantentypen).

Ausrichtung von Speicherbereichen

Die Ausrichtung von Speicherbereichen, also die Erhöhung des Adreßpegels auf eine bestimmte Grenze, ist abhängig vom Typ des Bereichs. Wenn ein Längenfaktor angegeben wurde, erfolgt in keinem Fall eine Ausrichtung.

Enthält der Operand mehr als eine Wertangabe, dann wird nur der erste Wert ausgerichtet.

Typ	Ausrichtung auf
C	Byte-Grenze
X	Byte-Grenze
B	Byte-Grenze
P	Byte-Grenze
Z	Byte-Grenze
F	Wort-Grenze
H	Halbwort-Grenze
E	Wort-Grenze
D	Doppelwort-Grenze
L	Doppelwort-Grenze
A	Wort-Grenze
Y	Halbwort-Grenze
S	Halbwort-Grenze
V	Wort-Grenze
Q	Wort-Grenze

Tabelle 4-12 Ausrichtung von Speicherbereichen in der DS-Anweisung

Programmierhinweise

1. Erzwungene Ausrichtung:
   Mit Hilfe einer DS-Anweisung können Speicherbereiche oder Konstanten, die normalerweise nicht ausgerichtet werden, auf Halbwort-, Wort- oder Doppelwortgrenze ausgerichtet werden. Dazu muß der jeweiligen DC- oder DS-Anweisung eine DS-Anweisung vorangestellt werden mit dem Wiederholungsfaktor Null und der entsprechenden Typangabe (z.B. H, F oder D).

2. Redefinieren von Speicherbereichen:
   Mit einer DS-Anweisung mit dem Wiederholungsfaktor Null wird einem Speicherbereich ein Name und ein Längenmerkmal zugeordnet. Mit weiteren folgenden DS- oder DC-Anweisungen kann dieser Bereich redefiniert werden, d.h., Felder, bzw. Konstanten innerhalb definiert und einzeln angesprochen werden (siehe Beispiele, (02)). Felder, die in einem solchen Bereich nicht definiert werden sollen, müssen mit einer DS-Anweisung mit Längenangabe oder mit einer ORG-Anweisung übersprungen werden.

3. Längenberechnung:
   Für Speicherbereiche, deren implizite Länge 1 byte beträgt (siehe Tabelle 4-8), kann man den Assembler das Längenmerkmal berechnen lassen, wenn man die entsprechenden Werte im Feld chrcon, bzw. datcon angibt. In diesem Fall darf kein Längenfaktor angegeben werden. Der Assembler berechnet an Hand der Werte die benötigte Länge.

4. Die DS-Anweisung reserviert einen Speicherbereich, belegt ihn jedoch nicht mit Nullen. Der Inhalt des reservierten Bereichs kann nicht vorhergesagt werden.

Beispiele

```
Name      | Operation | Operanden
----------+-----------+-----------------
FIELD1    | DS        | CL80              1 Feld, Längenmerkmal 80
FIELD2    | DS        | 4CL20             4 Felder, Längenmerkmal je 20
FIELD3    | DS        | 4C                4 Felder, Längenmerkmal je 1
FIELD4    | DS        | F                 1 Feld, ausgerichtet auf Wort-
          | .         |                   grenze, Längenmerkmal 4
          | .         |
          | DS        | 0F                ┐
AREA      | DS        | XL20              } (01)
          | .         |                   ┘
          | .         |
RDAREA    | DS        | 0CL50             ┐
          | DS        | CL4               │
PERSNUM   | DS        | CL6               │
NAME      | DS        | CL20              │
          | DS        | CL4               } (02)
DATE      | DS        | 0CL6              │
DAY       | DS        | CL2               │
MON       | DS        | CL2               │
YEA       | DS        | CL2               ┘
          | ORG       | RDAREA+L'RDAREA   (03)
```

(01) Durch die vorangestellte DS-Anweisung mit Wiederholungsfaktor Null und der Typangabe F wird der Bereich AREA auf Wortgrenze ausgerichtet.

(02) Für den Bereich RDAREA wird kein Speicherplatz reserviert, er erhält aber ein Längenmerkmal.
Die folgenden Anweisungen redefinieren den Bereich, wobei das Feld DATE nochmals unterteilt wird. Man kann sowohl die einzelnen Felder, als auch den Bereich als Ganzes ansprechen.

(03) Die ORG-Anweisung am Ende der Anweisungsfolge erhöht den Adreßpegel auf das Ende des Bereichs RDAREA, damit der gesamte Bereich reserviert ist.

>>>>> siehe auch DC-Anweisung

DXD Pseudoregister definieren

Funktion

Die DXD-Anweisung definiert ein Pseudoregister.

Format

Name	Operation	Operanden
name	DXD	[dup]typ[Ln][’val’]

name Name
dup Wiederholungsfaktor
dezimaler selbstdefinierender Wert oder positiver absoluter Ausdruck in Klammern
typ Typ des Pseudoregisters, ein einzelner Buchstabe
Ln Längenfaktor
n ist ein dezimaler selbstdefinierender Wert oder ein positiver absoluter Ausdruck in Klammern,
Maximalwert: dup*n ≤ 4095
val Wertangabe für das Pseudoregister

Hinweis

Es ist möglich, im Operanden der DXD-Anweisung das ganze Format eines Operanden der DC-Anweisung anzugeben. Diese zusätzlichen Angaben haben aber keine Auswirkung auf das Pseudoregister selbst.

Beschreibung

name ist der Name des Pseudoregisters. Der Wert des Namens ist die niedrigstwertige Adresse des Pseudoregisters. Dieser Wert wird erst beim Laden eingetragen. Bis dahin hat der Name einen vorläufigen Wert von Null.

dup gibt an, wie oft der im Operanden angegebene Bereich reserviert werden soll.

Ein **Wiederholungsfaktor mit dem Wert Null** ist erlaubt und hat folgende Wirkung: Es wird ein Pseudoregister mit der Länge Null definiert, das entsprechend seinem Typ ausgerichtet ist.

typ — Wenn kein Längenfaktor angegeben ist, bestimmt der Typ die Ausrichtung des Pseudoregisters und seine Länge.

mögliche Typangaben:

C	Zeichenbereiche
B	binäre Bereiche
P, Z	dezimale Bereiche
X	sedezimale Bereiche
F, H	Festpunktbereiche
E, D, L	Gleitpunktbereiche
A, Y, S, V, Q	Adreßbereiche

Ln — gibt die Länge des Pseudoregisters an und überschreibt die implizite Länge.
Bei Angabe eines Längenfaktors wird das Pseudoregister nicht ausgerichtet.
Mögliche Werte für den Längenfaktor und implizite Längen siehe DS-Anweisung.

val — Wert, der im Operanden angegeben werden kann.
Wenn kein Längenfaktor angegeben ist, berechnet der Assembler an Hand des angegebenen Wertes die benötigte Länge des Pseudoregisters.
Der angegebene Wert muß für den Typ des Pseudoregisters zulässig sein (siehe Beschreibung der Konstantentypen).

Programmierhinweise

1. Der Binder bestimmt die Reihenfolge, in der alle Pseudoregister aus allen zusammenzubindenden Moduln zu einem Psudoregistervektor zusammengefaßt werden.
2. Wenn ein Pseudoregister in mehreren Übersetzungseinheiten mit unterschiedlicher Länge und Ausrichtung definert ist, verwendet der Binder die jeweils stärksten Attribute als letztendlich gültige.
3. Die Pseudoregister können an beliebiger Stelle im Programm definiert werden. Es ist nicht nötig, daß die Definitionen aller Pseudoregister hintereinander aufgeführt sind.
4. Zur Zeit unterstützt der DLL die Verwendung von Pseudoregistern nicht, d.h., daß solche Programme mit TSOSLNK gebunden werden müssen und kein Nachlademechanismus in Anspruch genommen werden kann (siehe Bindelader-Starter Benutzerhandbuch [8]).

Beispiel siehe Anhang 11.4

>>>>> siehe auch DC-, DSECT- und CXD-Anweisung

DROP Basisadreßregister freigeben

Funktion

Die DROP-Anweisung bewirkt, daß ein bisher zugewiesenes Basisadreßregister wieder zur allgemeinen Verwendung zur Verfügung steht.

Format

Name	Operation	Operanden
[.sym]	DROP	[reg[,...]]

.sym Folgesymbol

reg Mehrzweckregister; positive absolute Ausdrücke, entweder
- Namen, denen ein absoluter Wert von 0 bis 15 zugewiesen wurde oder
- dezimale selbstdefinierende Werte von 0 bis 15

Beschreibung

reg bezeichnet die Basisadreßregister, die in einer USING-Anweisung zugewiesen wurden und jetzt wieder freigegeben werden. D.h., daß sie nicht mehr als Basisregister zur Verfügung stehen.

DROP mit leerem Operandenfeld gibt alle Basisadreßregister frei.

Programmierhinweise

1. Ein Register, das mit einer DROP-Anweisung gesperrt wurde, kann später wieder mit einer neuen USING-Anweisung für die Adressierung verfügbar gemacht werden.
2. Es ist nicht notwendig, eine DROP-Anweisung zu verwenden, wenn die Basisadresse mit einer USING-Anweisung geändert werden soll.

>>>>> siehe auch USING-Anweisung

DSECT Pseudoabschnitt definieren

Funktion

Die DSECT-Anweisung kennzeichnet den Beginn oder die Fortsetzung eines Pseudoabschnitts.

Format

Name	Operation	Operanden
name	DSECT	

name Name

Beschreibung

name kennzeichnet den Namen des Pseudoabschnitts.

Eine weitere DSECT-Anweisung mit demselben Namen kennzeichnet die Fortsetzung des Pseudoabschnitts.

Das Längenmerkmal von name, bzw. &par ist 1.

Durch die DSECT-Anweisung, die den Beginn des Pseudoabschnitts kennzeichnet, wird ein neuer Adreßpegel aufgebaut und ihm der Anfangswert Null zugewiesen.

Durch die Definition eines Pseudoabschnitts wird kein Speicherplatz reserviert.

Programmierhinweise

1. Nach der DSECT-Anweisung werden die Felder beschrieben, die durch den Pseudoabschnitt auf einen Hauptspeicherbereich projiziert werden sollen.
2. Die Namen, die in einem Pseudoabschnitt definiert sind, können als Operanden in Assemblerbefehlen benutzt werden. Dazu ist folgendes notwendig (siehe Beispiel):
   – Dem Assembler muß mit einer USING-Anweisung ein Basisadreßregister zur Verfügung gestellt werden, das ab der Anfangsadresse des Pseudoabschnitts wirksam sein soll.
   – Es muß sichergestellt sein, daß das Basisadreßregister beim Programmablauf mit der aktuellen Adresse des Speicherbereichs geladen ist, auf den der Pseudoabschnitt projiziert werden soll.
3. Ein Name, der in einem Pseudoabschnitt definiert ist, darf in einer Adreßkonstanten vom Typ A nur verwendet werden, wenn er mit einem anderen mit entgegengesetztem Vorzeichen im selben Pseudoabschnitt gepaart ist.
4. Mit der DSECT-Anweisung kann ein Pseudoregister definiert werden. Der Name der DSECT-Anweisung muß dann als Operand einer Adreßkonstanten vom Typ Q auftreten.

Beispiele

Das folgende Beispiel zeigt zwei getrennt übersetzte Programme, PROG1 und PROG2. PROG1 soll einen Satz einlesen, PROG2 soll Teile aus diesem Satz verarbeiten. Die Übergabe der Daten von PROG1 nach PROG2 erfolgt über einen mit einer DSECT spezifizierten Speicherbereich. In diesem Beispiel bleibt das Basisadreßregister gleich und der Speicherinhalt ändert sich. Es wird also eine Eingabe gelesen und mit überlagerter DSECT-Struktur verarbeitet.

```
Name        Operation   Operanden
PROG1       START
R0          EQU         0
R2          EQU         2
R3          EQU         3
R4          EQU         4
R14         EQU         14
R15         EQU         15
            .
            .
            BALR        R3,0
            USING       *,R3
            .
            .
READNEXT    RDATA       IN,ERR
            .
            .
            LA          R4,SEN              (01)
            L           R15,=V(PROG2)
            BALR        R14,R15
            .
            .
            B           READNEXT
ERR         TERM
            .
IN          DS          0CL84
            DS          CL4
SEN         DS          CL80
            .
            .
            END
            .
            .
            .
PROG2       START
            .
            .
            BALR        R2,R0
            USING       *,R2
            USING       BEG,R4              (02)
            .
            .
BEG         DSECT                           (03)
NR          DS          CL2
NAME        DS          CL2
STREET      DS          CL18                (04)
            .
            .
            END
```

(01) Die Anfangsadresse von SEN wird ins Register 4 geladen.
(02) Register 4 wird als Basisadreßregister für den Pseudoabschnitt zugewiesen und die Anfangsadresse des Pseudoabschnitts angegeben.
(03) Definition des Pseudoabschnitts.
(04) "Schablone", die über den Bereich SEN gelegt wird.

Im zweiten Beispiel soll eine Tabelle gelesen werden. Voraussetzung ist, daß die Tabelle vorher gefüllt worden ist und jedes Tabellenelement eine feste Satzstruktur hat.
Der Pseudoabschnitt entspricht einem Tabellenelement. die Tabelle wird gelesen durch Weiterschalten des Basisadreßregisters um die Elementlänge bis zum Tabellenende. In diesem Beispiel wird also das Basisadreßregister verändert, um andere Speicherbereiche zu adressieren.

```
Name      | Operation | Operanden
----------|-----------|-------------------
          | CSECT     |
R1        | EQU       | 1
R2        | EQU       | 2
R15       | EQU       | 15
          | .         |
          | .         |
          | EXTRN     | TABEND
          | EXTRN     | TAB
          | USING     | *,R15
          | USING     | TABELE,R1
          | LA        | R1,TAB
          | LA        | R2,TABEND
LOOP      | MVC       | OUT,STREET
          | .         |
          | .         |
          | LA        | R1,LTABELE(0,R1)
          | CR        | R1,R2
          | BL        | LOOP
          | .         |
          | .         |
TABELE    | DSECT     |
NR        | DS        | CL2
NAME      | DS        | CL2
STREET    | DS        | CL18
LTABELE   | EQU       | *-TABELE
          | .         |
          | .         |
```

>>>>> siehe auch CSECT- und XDSEC-Anweisung

EJECT Seitenvorschub

Funktion

Die EJECT-Anweisung bewirkt einen Seitenvorschub auf dem Übersetzungsprotokoll.

Format

Name	Operation	Operanden
[.sym]	EJECT	

.sym Folgesymbol

Beschreibung

Die der EJECT-Anweisung folgende Instruktion im Programm erscheint im Übersetzungsprotokoll auf einer neuen Seite. Steht eine solche Anweisung bereits am Beginn einer neuen Seite, wird die EJECT-Anweisung ignoriert.

Zwei EJECT-Anweisungen erzeugen eine Leerseite, etc.

>>>>> siehe auch SPACE-, TITLE- und PRINT-Anweisung

END Übersetzungsende

Funktion

Die END-Anweisung kennzeichnet das Ende einer Übersetzungseinheit und kann die Adresse des ersten auszuführenden Befehls des Programms angeben.

Format

Name	Operation	Operanden
[.sym]	END	[ausdr]

.sym Folgesymbol
ausdr positiver relativer Ausdruck

Beschreibung

Der Operand ausdr in der END-Anweisung gibt die Adresse des Befehls an, mit dem nach dem Laden des Programms die Ausführung beginnen soll.

Programmierhinweise

1. Die END-Anweisung muß immer die letzte Instruktion in einem Programm sein.
2. ausdr kann eine Adresse in einem getrennt übersetzten Programm bezeichnen. In diesem Fall muß ausdr in einer Adreßkonstante vom Typ V oder in einer EXTRN-Anweisung gekennzeichnet sein (siehe zweites Beispiel).
3. Wird eine END-Anweisung durch einen Makro generiert, so wird der Rest des Quellprogramms übersprungen, d.h. dieser Rest wird nicht mehr übersetzt (auf diese Weise ist damit auch keine Mehrfachübersetzung möglich).

Beispiele

```
Name        | Operation | Operanden
------------+-----------+-----------
PROG        | CSECT     |
R0          | EQU       | 0
R3          | EQU       | 3
            | .         |
            | .         |
BEGIN       | BALR      | R3,R0
            | USING     | *,R3
            | .         |
            | .         |
            | .         |
            | END       | BEGIN
```

Das folgende Beispiel zeigt eine END-Anweisung mit einer externen Adresse, die als V-Konstante gekennzeichnet ist.

```
Name        | Operation | Operanden
------------+-----------+-----------
PROG        | CSECT     |
            | .         |
            | .         |
            | DC        | V(STARTMO)
            | .         |
            | .         |
            | END       | STARTMO
```

>>>>> siehe auch START-, CSECT-, DSECT- und XDSEC-Anweisung

ENTRY ENTRY-Adresse kennzeichnen

Funktion

Die ENTRY-Anweisung kennzeichnet eine symbolische Adresse, die in einer Übersetzungseinheit definiert ist und von einer anderen aus angesprochen werden soll.

Format

Name	Operation	Operanden
[.sym]	ENTRY	name[,...]

.sym Folgesymbol
name Name

Beschreibung

name kennzeichnet eine Adresse, die eine andere Übersetzungseinheit als Sprungziel oder als Datenadresse benutzen kann.

name ist durch die ENTRY-Anweisung noch nicht definiert. Das Längenmerkmal von name wird daher von der Instruktion festgelegt, die name definiert.

Programmierhinweise

1. Der Name eines Programmabschnitts muß nicht durch eine ENTRY-Anweisung gekennzeichnet sein, damit ein anderes Programm darauf Bezug nehmen kann.
2. Eine ENTRY-Anweisung darf keine Namen enthalten, die in einem Pseudoabschnitt definiert sind.

Beispiel siehe EXTRN-Anweisung

>>>>> siehe auch EXTRN- und WXTRN-Anweisung

EXTRN EXTRN-Adresse kennzeichnen

Funktion

Die EXTRN-Anweisung kennzeichnet eine symbolische Adresse, die in einer Übersetzungseinheit angesprochen wird und in einer anderen definiert ist.

Format

Name	Operation	Operanden
[.sym]	EXTRN	name[,...]

.sym Folgesymbol
name Name

Beschreibung

name kennzeichnet eine Adresse, die in einer anderen Übersetzungseinheit als Einsprungstelle definiert ist.

Aufgrund der EXTRN-Anweisung wird für den Binder Information über den EXTRN-Verweis abgesetzt.

name ist durch die EXTRN-Anweisung für diese Übersetzungseinheit definiert. Der Assembler weist ein Längenmerkmal 1 und einen Wert 0 zu.

Programmierhinweise

1. Namen, die in einer EXTRN-Anweisung definiert sind, dürfen nicht als Namen von Instruktionen im gleichen Programm auftreten.
2. Wenn Namen, die als EXTRN definiert wurden, in arithmetischen Ausdrücken verwendet werden, dürfen sie nicht paarweise auftreten.

Beispiel zu EXTRN und ENTRY

Das folgende Beispiel zeigt die Verknüpfung von zwei unabhängigen Übersetzungseinheiten mit Hilfe der ENTRY- und EXTRN-Anweisung. In PROG1 ist die Adresse IN als extern gekennzeichnet. PROG2 enthält IN als Sprungziel.

```
Name     | Operation | Operanden
---------+-----------+-----------
PROG1    | START     |
R0       | EQU       | 0
R3       | EQU       | 3
R4       | EQU       | 4
R10      | EQU       | 10
R15      | EQU       | 15
         | .         |
         | .         |
         | EXTRN     | IN
         | BALR      | R10,R0
         | USING     | *,R10
         | .         |
         | .         |
         | L         | R15,=A(IN)
         | BALR      | R14,R15
         | .         |
         | .         |
         | END       | PROG1
         | .         |
         | .         |
         | .         |
PROG2    | START     |
R0       | EQU       | 0
R3       | EQU       | 3
R4       | EQU       | 4
R15      | EQU       | 15
         | ENTRY     | IN
         | .         |
         | .         |
IN       | EQU       | *
         | USING     | *,R15
         | .         |
         | .         |
         | BR        | R14
         | END       | PROG2
```

>>>>> siehe auch ENTRY- und WXTRN-Anweisung

EQU Gleichsetzen

Funktion

Die EQU-Anweisung weist einem Namen den Wert und die Eigenschaften des Ausdrucks zu, der im Operandeneintrag steht.

Format

Name	Operation	Operanden
name	EQU	ausdr[,[len][,typ]]

name Name
ausdr absoluter oder relativer Ausdruck
len positiver absoluter Ausdruck, Wertebereich 0 bis 2^{24}-1
typ selbstdefinierender Wert, maximal 1 Byte lang

Beschreibung

name Dem so bezeichneten Feld wird der Wert von ausdr zugewiesen. Der Wert des Namens ist relativ oder absolut, je nachdem, ob ausdr einen relativen oder absoluten Wert hat.

name erhält das gleiche Längenmerkmal wie ausdr.

Wenn ausdr ein arithmetischer Ausdruck ist, erhält name das Längenmerkmal und das Typenmerkmal des am weitesten links stehenden Namen im Operanden. Besteht der Ausdruck im Operandenfeld aus einem Stern (*) oder aus einem selbstdefinierenden Wert, ist das Längenmerkmal von name 1, und das Typenmerkmal U.

ausdr In ausdr verwendete Namen müssen nicht vorher definiert sein.

len Wenn man len angibt, wird name dieser Wert als Längenmerkmal zugewiesen.
Wenn das Längenmerkmal in der Makrosprache ausgewertet werden soll, darf len nur ein selbstdefinierender Wert sein. Bei Angabe eines Ausdrucks oder eines symbolischen Parameters für len wird während der Makroauflösung der Standardwert 1 zugewiesen.

typ Wenn typ kein selbstdefinierender Wert ist, hat der Name das Typenmerkmal U (undefiniert). Das Typenmerkmal kann nur in der Makroverarbeitung ausgewertet werden.
Die Angabe kann als selbstdefinierender dezimaler, sedezimaler, binärer oder als Zeichenwert erfolgen.

Programmierhinweise

1. Die EQU-Anweisung wird häufig als Hilfsmittel für die strukturierte Programmierung eingesetzt,z.B. durch
   - Zuweisen von Namen für Registernummern,
   - Trennen von Sprungadressen und Befehlen durch Zuweisen des Sprungziels zum aktuellen Adreßpegel und
   - Zuweisen von Namen für häufig verwendete Ausdrücke.

   Beispiele

```
Name      | Operation | Operanden
----------|-----------|-----------------------
REG1      | EQU       | 1                      (01)
MARK      | EQU       | *                      (02)
TERM1     | EQU       | A-B*C/2          ┐
TERM2     | EQU       | B'101011111'     } (03)
TERM3     | EQU       | -10              ┘
```

   (01) Dem Namen REG1 wird der Wert 1 zugewiesen. REG1 kann dann als Registerbezeichnung verwendet werden.
   (02) Dem Namen MARK wird der Wert des aktuellen Adreßpegels zugewiesen.
   (03) Den Namen TERM1, TERM2 und TERM3 wird der Wert des Ausdrucks im Operanden zugewiesen.

2. Ausdrücke, deren Wert man bei der Programmerstellung noch nicht kennt, können ebenfalls Namen zugeordnet werden. Dem Namen wird dann der vom Assembler berechnete Wert des Ausdrucks im Operanden zugewiesen.

3. Bei der Makroauflösung durch den Assembler ist die EQU-Anweisung noch nicht wirksam.

   Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------
REG1      | EQU       | 1
          | MAC       | REG1
```

   Diese Folge von EQU-Anweisung und Makroaufruf bewirkt eine Generierung des Makro MAC mit der Zeichenfolge REG1 als Operand und nicht mit Register 1.

4. Die explizite Längenangabe in der EQU-Anweisung wird häufig verwendet, um beim Zugriff einen größeren Bereich ansprechen zu können.

   Beispiele

```
Name      | Operation | Operanden
----------|-----------|-----------------
LOOP1     | EQU       | *                (01)
LOOP2     | EQU       | *,4              (02)
LOOP3     | EQU       | *,L'LOOP2        (03)
```

   (01) LOOP1 hat den Wert des aktuellen Adreßpegels und das Längenmerkmal 1.
   (02), (03)
   Bei LOOP2 und LOOP3 wird das Längenmerkmal auf 4 byte gesetzt.

5. Mit der expliziten Typangabe wird das ursprüngliche Typenmerkmal verändert.

   Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------
TERM1     | EQU       | *,,C'A'          ] Beiden Namen wird der Typ A
TERM2     | EQU       | *,,X'C1'         ] (Adreßkonstante) zugewiesen
```

ICTL Eingabeformat steuern

Funktion

Mit der ICTL-Anweisung kann die Voreinstellung für die Anfangs-, End- und Fortsetzungsspalte von Quellenprogrammanweisungen geändert werden.

Format

Name	Operation	Operanden
	ICTL	a[,e[,f]]

a dezimaler selbstdefinierender Wert von 1 bis 40
e dezimaler selbstdefinierender Wert von 41 bis 80
f dezimaler selbstdefinierender Wert von 1 bis 40

Beschreibung

a legt die Anfangsspalte der Quellprogrammanweisungen fest.

e legt die Endspalte der Quellprogrammanweisung fest. Wird e nicht angegeben, dann gilt die Voreinstellung.

Ein Wert von 80 für e bedeutet, daß keine Fortsetzungszeilen auftreten.

f legt die Fortsetzungsspalte für die Quellprogrammanweisungen fest. f muß gleich oder größer als a sein.

Weglassen von f bedeutet, daß keine Fortsetzungszeilen auftreten.

Programmierhinweise

1. Die Voreinstellung der Anfangs-, End- und Fortsetzungsspalte sind die Spalten 1, 71 und 16 (siehe 2.2, Instruktionen).
2. Die ICTL-Anweisung darf in einer Übersetzungseinheit nur einmal vorkommen und muß die erste Anweisung im Programm sein.
3. Die ICTL-Anweisung hat keine Auswirkung auf eingelesene Makros.
4. Der Operationscode der ICTL-Anweisung darf nicht mit Hilfe von variablen Parametern generiert werden.
5. Statt über die ICTL-Anweisung kann das Eingabeformat auch über eine SDF-Option gesteuert werden (siehe ASSEMBH, Benutzerhandbuch [1]).

LTORG Literalbereich definieren

Funktion

Die LTORG-Anweisung definiert einen Literalbereich und legt seine Lage fest.

Format

Name	Operation	Operanden
[{name / .sym}]	LTORG	

name Name
.sym Folgesymbol

Beschreibung

Die LTORG-Anweisung bewirkt, daß ab der nächsten Doppelwortgrenze ein Literalbereich angelegt wird.

In diesem Literalbereich werden alle fehlerfreien Literale abgelegt, die seit der vorherigen LTORG-Anweisung oder seit Programmanfang aufgetreten sind.

Die Literale werden innerhalb des Literalbereichs entsprechend ihrem Typ und ihrer Länge ausgerichtet. Sie werden in der Reihenfolge Doppelwort, Wort, Halbwort, Byte abgelegt.

Alle Literale, die nach der letzten LTORG-Anweisung folgen, werden am Ende des ersten Programmabschnitts abgelegt.

Der Wert von name bzw. .sym ist die Adresse des ersten Bytes des Literalbereichs. name erhält ein Längenmerkmal von 1.

Programmierhinweise

1. Enthält ein Programm keine LTORG-Anweisung, dann werden alle Literale in einen Literalbereich am Ende des ersten Programmabschnitts abgelegt. In diesem Fall muß der erste Programmabschnitt immer adressierbar sein.

   Der Literalbereich wird dann im Übersetzungsprotokoll nach der END-Anweisung aufgeführt.

2. Eine LTORG-Anweisung am Ende jedes Programmabschnitts sorgt dafür, daß alle Literale des jeweiligen Abschnitts immer adressierbar sind.

3. Literale, die im Bereich einer LTORG-Anweisung mehrfach vorkommen, werden nur einmal abgelegt. Eine Ausnahme sind hier Literale, die einen Verweis auf den Adreßpegel enthalten. Diese Literale werden mit dem jeweils aktuellen Wert des Adreßpegels einzeln abgelegt.

OPSYN Mnemotechnischen Operationscode zuweisen

Funktion

Die OPSYN-Anweisung weist einem Namen die Eigenschaften des mnemotechnischen Operationscodes oder des Makronamens im Operanden zu oder sorgt dafür, daß er diese Eigenschaften verliert.

Format

Name	Operation	Operanden
name	OPSYN	[code]

name
- Name oder
- mnemotechnischer Operationscode einer Assembleranweisung, eines Assemblerbefehls oder einer Makroanweisung oder
- Makroname

code
- mnemotechnischer Operationscode einer Assembleranweisung, eines Assemblerbefehls oder einer Makroanweisung,
- Makroname oder
- Name

Beschreibung

Dem Namenseintrag werden die Eigenschaften des Eintrags im Operandenfeld zugewiesen. Die Eigenschaften des Operandeneintrags bleiben unverändert.

Ein leeres Operandenfeld bewirkt, daß name die Eigenschaften eines mnemotechnischen Operationscodes verliert (siehe Beispiel (02)/(03)).

Der gleiche Eintrag im Namensfeld und im Operandenfeld bewirkt, daß name wieder seine ursprünglichen Eigenschaften annimmt. Ein gleicher Eintrag im Namens- und im Operandenfeld ist nur für mnemotechnische Operationscodes von Assembleranweisungen und Assemblerbefehlen erlaubt.

Programmierhinweise

1. Die OPSYN-Anweisung kann an jeder Stelle im Quellprogrammtext, auch innerhalb von Makros, stehen.
2. Es gilt immer die zuletzt durchlaufene OPSYN-Anweisung bis zur nächsten OPSYN-Anweisung. Eine OPSYN-Anweisung, die in einem Makro generiert wurde, gilt nicht nur für den Makro selbst, sondern auch für alle folgenden Instruktionen. Wird der Makro oder die OPSYN-Anweisung übersprungen und nicht durchlaufen, dann tritt die OPSYN-Anweisung auch nicht in Kraft.
3. Wenn name der mnemotechnische Operationscode eines Assemblerbefehls oder einer Assembleranweisung ist, dann wird dieser Operationscode umdefiniert. Ist das Operandenfeld leer, dann wird dieser Operationscode im folgenden nicht mehr als Assemblerbefehl oder -anweisung erkannt und wie ein Makroaufruf behandelt.
4. Bei der Bearbeitung von Operationscodes, die in einer Makrodefinition liegen, ist folgendes zu beachten:
   - Bei Bibliotheksmakros werden alle Makroanweisungen, sowie die COPY- und die REPRO-Anweisung entsprechend der OPSYN-Anweisung bearbeitet, die beim ersten Aufruf des Makro gültig war.
   - Bei einer Makrodefinition im Quellprogramm werden alle Makroanweisungen, sowie die COPY- und die REPRO-Anweisung entsprechend der OPSYN-Anweisung bearbeitet, die beim Lesen der Makrodefinition gültig war.
   - Alle übrigen Instruktionen werden bei Bibliotheksmakros und bei Makros im Quellprogramm entsprechend der OPSYN-Anweisung bearbeitet, die direkt vor der jeweiligen Instruktion durchlaufen wird.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------------
EOP       | OPSYN     | TERM              (01)
          | .         |
          | .         |
          | EOP       |
          | .         |
          | .         |
STORE     | OPSYN     | STH               (02)
STH       | OPSYN     |                   (03)
ABC       | OPSYN     | STORE             (04)
          | .         |
          | .         |
STORE     | OPSYN     |                   (05)
STH       | OPSYN     | STH               (06)
```

(01) Der Zeichenfolge EOP werden die Eigenschaften des TERM-Makro zugewiesen. EOP im Operationsfeld bedeutet im folgenden den Aufruf des Makro TERM.

(02) Redefiniert den Assemblerbefehl STH, STORE wird im folgenden als STH interpretiert.

(03) STH wird ab sofort nicht mehr als Assemblerbefehl erkannt.

(04) ABC wird zusätzlich und mit gleicher Wirkung wie STORE eingeführt.

(05) STORE verliert ab sofort seine Eigenschaften als Assemblerbefehl, ABC definiert aber weiterhin den Assemblerbefehl STH.

(06) Läßt den Assemblerbefehl STH wieder in seiner ursprünglichen Bedeutung zu.

ORG Adreßpegel setzen

Funktion

Mit der ORG-Anweisung kann der augenblickliche Wert des Adreßpegels verändert werden.

Format

Name	Operation	Operanden
[{name / .sym}]	ORG	[ausdr]

name Name
.sym Folgesymbol
ausdr relativer Ausdruck

Beschreibung

ausdr bezeichnet den Wert, auf den der Adreßpegel gesetzt werden soll. Die ORG-Anweisung ohne Operand setzt den Adreßpegel hinter die höchste bisher angesprochene Speicherstelle des Programmabschnitts.

Die Namen, die in ausdr verwendet werden, müssen vorher definiert sein.

Der Wert von ausdr muß eine relative Adresse im aktuellen Programmabschnitt sein, d.h., ausdr darf keine Namen enthalten, die in einem anderen Programmabschnitt definiert sind.

Der Endwert der Berechnung von ausdr darf maximal 2^{24}-1 sein.

Programmierhinweise

1. ausdr darf keinen Wert annehmen, der vor dem Beginn des Programmabschnitts liegt, in dem die ORG-Anweisung steht.
2. Wenn vorhergehende ORG-Anweisungen den Adreßpegel zurückgesetzt haben, kann eine ORG-Anweisung ohne Operand verwendet werden, um den Adreßpegel wieder hinter die höchste bisher angesprochene Speicherstelle zu setzen.
3. Beim Zurücksetzen des Adreßpegels dürfen keine EXTRN-Verweise, keine CXD-Anweisungen und keine A-, Q-,V-und Y-Konstanten überschrieben werden (also keine Instruktionen, die dazu führen, daß RLD-Informationen abgelegt werden).

Beispiel

Das folgende Beispiel zeigt die Anwendung der ORG-Anweisung bei der Definition eines Speicherbereichs. Die ORG-Anweisung am Ende der Anweisungsfolge erhöht den Adreßpegel auf das Ende des Bereichs RDAREA.

```
Name      | Operation | Operanden
----------+-----------+----------
RDAREA    | DS        | CL30
          | ORG       | RDAREA
          | DS        | CL4
NAME      | DS        | CL10
          | DS        | CL4
PERSN     | DS        | CL10
          | .         |
          | .         |
          | ORG       |
```

PRINT Protokollinhalt steuern

Funktion

Die PRINT-Anweisung bestimmt, welche Teile des Übersetzungsprotokolls ausgedruckt werden sollen.

Format

Name	Operation	Operanden
[.sym]	PRINT	[{BASE / NOBASE}] [{CODE / NOCODE}] [{COPY / NOCOPY}] {[{DATA / NODATA}] [{GEN / NOGEN}] [{ON / OFF}] [{SINGLE / DOUBLE}]}

.sym Folgesymbol

Operanden 1 bis 7 Operanden in beliebiger Reihenfolge; wenn mehr als ein Operand angegeben wird, müssen sie durch Komma getrennt werden.

Beschreibung

BASE Nach jeder USING- oder DROP-Anweisung wird der adressierbare Bereich für die Register ausgedruckt, die mit USING als Basisregister eingeführt wurden. Zusätzlich wird der Bemerkungseintrag aus der USING-Anweisung 19 Zeichen lang ausgegeben.

NOBASE Die adressierbaren Bereiche für die Basisregister werden nicht ausgedruckt.

CODE	Bei Instruktionen, die durch Makros generiert wurden, wird die Wirkung von PRINT NOGEN eingeschränkt: Der Text der Quellzeile wird unterdrückt, der generierte Code wird jedoch ausgedruckt.
NOCODE	Die volle Wirkung von PRINT NOGEN bleibt erhalten.
COPY	Kopierte Anweisungen werden protokolliert
NOCOPY	Kopierte Anweisungen werden nicht protokolliert.
DATA	Konstanten werden im Übersetzungsprotokoll vollständig ausgedruckt.
NODATA	Nur die ersten 8 Bytes von Konstanten werden im Übersetzungsprotokoll ausgedruckt.
GEN	Alle Instruktionen, die durch Makroaufrufe generiert werden, werden protokolliert.
NOGEN	Durch Makroaufrufe generierte Instruktionen werden nicht protokolliert. Meldungen, die von MNOTE-Anweisungen erzeugt werden, werden jedoch ausgedruckt.
ON	Das Protokoll wird ab hier ausgedruckt. PRINT ON ist die letzte Instruktion, die nicht protokolliert wird.
OFF	Das Protokoll wird ab hier nicht ausgedruckt. PRINT OFF ist die letzte Instruktion, die protokolliert wird.
SINGLE	Einfacher Zeilenabstand im Übersetzungsprotokoll
DOUBLE	Doppelter Zeilenabstand im Übersetzungsprotokoll

Programmierhinweis

Ein Programm kann beliebig viele PRINT-Anweisungen enthalten. Die Bedingungen, die von einer PRINT-Anweisung gesetzt wurden, gelten solange, bis sie durch eine weitere PRINT-Anweisung geändert werden. Bis zur ersten PRINT-Anweisung gelten die Standardwerte.

>>>>> siehe auch SPACE-, EJECT-, TITLE-, STACK- und UNSTCK-Anweisung

PUNCH Text in Objektmodul kopieren

Funktion

Die PUNCH-Anweisung gibt die Zeichen, die im Operanden angegeben sind, unverarbeitet in den Objektmodul aus.

Format

Name	Operation	Operanden
[.sym]	PUNCH	'text'

.sym Folgesymbol
text 1 bis 80 Zeichen

Beschreibung

Der Assembler legt die Daten, die in text dargestellt sind, unverarbeitet im Objektmodul ab.

Das erste Zeichen von text steht in der ersten Spalte des Ausgabeformates.

In den Objektmodul wird weder eine laufende Nummer noch eine Identifikation ausgegeben.

Programmierhinweise

1. Für jedes Hochkomma und jedes &-Zeichen, das im Objektmodul stehen soll, müssen zwei entsprechende Zeichen in text angegeben werden. Die beiden Zeichen werden jeweils nur als eines gezählt.
2. PUNCH-Anweisungen können an beliebiger Stelle im Programm oder in der Makrodefinition stehen.
3. Durch die PUNCH-Anweisung kann der Objektmodul unbrauchbar werden.
4. Die Leistung der PUNCH-Anweisung wird bei der Modulausgabe im LLM-Format nicht mehr unterstützt. Dies wird durch eine Fehlermeldung mit dem Gewicht 'Warnung' angezeigt.

>>>>> siehe auch REPRO-Anweisung

REPRO Folgezeile in Objektmodul kopieren

Funktion

Die REPRO-Anweisung gibt die nachfolgende Quellprogrammzeile unverarbeitet in den Objektmodul aus.

Format

Name	Operation	Operanden
[.sym]	REPRO	

.sym Folgesymbol

Beschreibung

Der Assembler legt die Quellprogrammzeile, die auf die REPRO-Anweisung folgt, unverändert im Objektmodul ab.

Das erste Zeichen in dieser Zeile steht in der ersten Spalte des Ausgabeformats.

In den Objektmodul wird weder eine laufende Nummer noch eine Identifikation ausgegeben.

Jede REPRO-Anweisung erzeugt eine Zeile im Objektmodul.

Programmierhinweise

1. Der Operationscode der REPRO-Anweisung darf nicht mit Hilfe von variablen Parametern generiert werden.
2. Durch die REPRO-Anweisung kann der Objektmodul unbrauchbar werden.
3. Die Leistung der REPRO-Anweisung wird bei der Modulausgabe im LLM-Format nicht mehr unterstützt. Dies wird durch eine Fehlermeldung mit dem Gewicht 'Warnung' angezeigt.

>>>>> siehe auch PUNCH-Anweisung

RMODE Ladeattribut zuordnen

Funktion

Die RMODE-Anweisung ordnet einem Programmabschnitt ein Ladeattribut zu.

Format

Name	Operation	Operanden
[{name / .sym}]	RMODE	{24 / ANY}

name Name
.sym Folgesymbol

Beschreibung

name bezieht sich auf einen gleichnamigen Programmabschnitt und muß mit dem Namen einer START-, CSECT- oder COM-Anweisung übereinstimmen.

Bei leerem Namensfeld bezieht sich die RMODE-Anweisung auf einen unbenannten Programmabschnitt.

.sym Ein Folgesymbol ist gleichbedeutend mit einem leeren Namensfeld.

24 Dem Programmabschnitt wird das Ladeattribut 24 zugeordnet. D.h., daß dieser Abschnitt nur unterhalb von 16 MB geladen werden darf.

ANY Das Ladeattribut des Programmabschnitts kann 24 oder 31 sein. D.h., daß dieser Abschnitt sowohl unterhalb als auch oberhalb von 16 MB geladen werden darf.

Zu ’Ladeattribut’ siehe Abschnitt 3.3.3, sowie das Handbuch Einführung in die XS-Programmierung [7].

Falls ein Lademodul mehrere Abschnitte mit verschiedenen Ladeattributen enthält, trifft das Binder-Lader-System die Auswahl eines gemeinsamen Ladeattributs für den gesamten Lademodul (siehe Binder und Lader, Beschreibung [8]).

Die Information über das Ladeattribut eines Programmabschnitts wird in den ESD-Satz des Objektmoduls übergeben.

Programmierhinweise

1. Die RMODE-Anweisung kann an beliebiger Stelle im Quellprogramm stehen. Das Quellprogramm darf beliebig viele RMODE-Anweisungen enthalten, wobei aber ein spezifizierter Name nur einmal auftreten darf.
2. Für einen unbenannten gemeinsamen Hilfsabschnitt (siehe COM-Anweisung) darf keine RMODE-Anweisung gesetzt werden.
3. Falls keine Spezialfunktionen erbracht werden sollen, sollten einem Programmabschnitt die Attribute AMODE = ANY und RMODE = ANY zugewiesen werden.

Kombinationen von AMODE und RMODE

Wenn für einen Programmabschnitt die AMODE-Anweisung gesetzt wurde, sind die folgenden Kombinationen mit der RMODE-Anweisung für den gleichen Programmabschnitt möglich:

AMODE 24 mit RMODE 24
AMODE 31 mit RMODE 24 oder RMODE ANY
AMODE ANY mit RMODE 24 oder RMODE ANY

Voreinstellung

Folgende Voreinstellungen gelten, wenn für einen Programmabschnitt entweder AMODE oder RMODE oder beide nicht gesetzt wurden:

angegeben	Voreinstellung
weder AMODE noch RMODE	AMODE 24 und RMODE 24
AMODE 24	RMODE 24
AMODE 31	RMODE 24
AMODE ANY	RMODE 24
RMODE 24	AMODE 24
RMODE ANY	AMODE 31

Tabelle 4-13 Voreinstellung für AMODE, bzw. RMODE

>>>>> siehe auch AMODE-Anweisung

SPACE Zeilenvorschub

Funktion

Mit der SPACE-Anweisung können Leerzeilen in das Übersetzungsprotokoll eingefügt werden.

Format

Name	Operation	Operanden
[.sym]	SPACE	[nr]

.sym Folgesymbol
nr dezimaler selbstdefinierender Wert

Beschreibung

nr gibt die Anzahl der Leerzeilen an, die nach der SPACE-Anweisung im Übersetzungsprotokoll erscheinen sollen. Ein leerer Operandeneintrag bewirkt einen Vorschub um eine Zeile.

Ist nr größer als die Zahl der auf dieser Seite noch verbleibenden Zeilen, dann bewirkt die SPACE-Anweisung einen Seitenvorschub.

>>>>> siehe auch EJECT-, PRINT- und TITLE-Anweisung

STACK USING- oder PRINT-Status sichern

Funktion

Mit der STACK-Anweisung kann der aktuelle USING-Status (d.h., die Basisregister und der zugehörige Adreßbereich) und/oder der aktuelle Stand der PRINT-Parameter gesichert werden.

Format

Name	Operation	Operanden
[.sym]	STACK	{PRINT[,...][,USING[,...]]} {USING[,...][,PRINT[,...]]}

.sym Folgesymbol

Programmierhinweise

1. Die STACK-Anweisung verändert weder den aktuellen USING-Status, noch die aktuellen PRINT-Parameter.
2. Die PRINT-Parameter, bzw. der USING-Status wird in einem "last-in-firstout-Verfahren" gespeichert. D.h., die letzte gespeicherte Information wird von der ersten entsprechenden UNSTK-Anweisung als erste wieder abgerufen.
3. In STACK-Anweisungen darf der Operand PRINT, bzw. USING maximal viermal nacheinander auftreten, bevor die erste UNSTK-Anweisung mit dem Operanden PRINT, bzw. USING folgen muß.

Beispiel

Das folgende Beispiel zeigt eine mögliche Anwendung der STACK USING-Anweisung. Die STACK-Anweisung am Beginn des Unterprogramms macht es möglich, innerhalb des Unterprogramms das Register 3 anders zu verwenden, ohne daß zuvor eine DROP-Anweisung nötig war. UNSTK am Ende des Unterprogramms setzt den USING-Status zurück auf die alten Werte, d.h., Register 3 ist wieder als Basisregister bekannt, Register 5 entsprechend dem vorherigen Status unbekannt.

```
Name     | Operation | Operanden
---------|-----------|-----------
PROG1    | START     |
R0       | EQU       | 0
R3       | EQU       | 3
R4       | EQU       | 4
R5       | EQU       | 5
         | .         |
         | .         |
         | BALR      | R3,0
         | USING     | *,R3
         | .         |
         | .         |
         | BAL       | R4,PROG2
         | .         |
         | .         |
         | TERM      |
         | .         |
         | .         |
PROG2    | EQU       | *
         | ST        | R3,SAFE
         | STACK     | USING
         | BALR      | R5,0
         | USING     | *,R5
         | L         | R3,NUM
         | AR        | R3,VAL
         | ST        | R3,NUM2
         | L         | R3,SAFE
         | UNSTK     | USING
         | BR        | R4
         | .         |
         | .         |
NUM      | DC        | F'1234'
VAL      | DC        | F'55'
NUM2     | DS        | F
SAFE     | DS        | F
         | .         |
         | .         |
```

>>>>> siehe auch UNSTK-Anweisung

START Programmanfang definieren

Funktion

Die START-Anweisung kennzeichnet den Beginn einer Übersetzungseinheit, ordnet dem Programm bzw. dem ersten Programmabschnitt eines Programms, einen Namen zu und setzt den Adreßpegel auf einen Anfangswert.

Format

Name	Operation	Operanden
[{name / .sym}]	START	[dec[,typ][,...]]

name Name
.sym Folgesymbol
dec selbstdefinierender Wert, der durch 8 teilbar sein sollte; Maximalwert: FFFFF8
typ Merkmalkennzeichnung für Programmabschnitte (siehe 3.3.3, Merkmale von Programmabschnitten)

Beschreibung

name kennzeichnet den Namen des Programms, bzw. den Namen des ersten Programmabschnitts.

Eine CSECT-Anweisung mit demselben Namen kennzeichnet die Fortsetzung des ersten Programmabschnitts.

Eine START-Anweisung ohne Namen kennzeichnet einen unbenannten Programmabschnitt.

Das Längenmerkmal des Namens ist 1.

dec dec bezeichnet den Anfangswert für den Adreßpegel des Programms.

Ist dec nicht durch 8 teilbar, dann wird der Adreßpegel auf die nächste Doppelwortgrenze gesetzt.

Ist dec nicht angegeben, setzt der Assembler den Anfangswert des Adreßpegels auf Null.

typ Bezeichnet Merkmale, die für das Programm, bzw. den ersten Programmabschnitt gelten sollen (siehe 3.3.3).

Programmierhinweise

1. Der START-Anweisung darf keine Instruktion vorausgehen, die den Adreßpegel benutzt oder verändert.
2. Die START-Anweisung kann durch die CSECT-Anweisung zur Bezeichnung des ersten Programmabschnitts ersetzt werden.
3. Alle Instruktionen, die zwischen der START-Anweisung und der nächsten CSECT-, bzw. DSECT-Anweisung stehen, gehören zum ersten Programmabschnitt. Der erste Programmabschnitt kann an anderer Stelle im Programm durch eine CSECT-Anweisung mit demselben Namen fortgesetzt werden.
4. Wenn eine oder mehrere Merkmalkennzeichnungen angegeben werden, darf der Operand dec nicht weggelassen werden.
5. Soll ein Assembler-Quellprogramm mit AID getestet werden, muß die START-Anweisung einen Namen haben. Für Assembler-Programme mit einem unbenannten ersten Programmabschnitt wird keine LSD-Information abgelegt (siehe AID, Testen von ASSEMBH-Programmen [2]).

>>>>> siehe auch END- und CSECT-Anweisung

TITLE Protokollüberschrift

Funktion

Die TITLE-Anweisung erzeugt die Seitenüberschriften des Übersetzungsprotokolls.

Format

Name	Operation	Operanden
[{name / .sym}]	TITLE	'text'

name Name, maximal vier alphanumerische Zeichen
.sym Folgesymbol
text 1 bis 97 Zeichen

Beschreibung

text bildet die Seitenüberschrift des Übersetzungsprotokolls. Jede neue TITLE-Anweisung in einem Programm bewirkt einen Seitenvorschub und die Ausgabe der neuen Überschrift am Kopf der folgenden Seiten.

Programmierhinweise

1. Wenn eine Seitenüberschrift ein &-Zeichen oder ein Hochkomma enthalten soll, müssen sie im text-Feld durch zwei &-Zeichen bzw. zwei Hochkommas dargestellt werden. Die zwei Zeichen werden jeweils als ein Zeichen gezählt und ausgedruckt.
2. Nur jeweils die erste TITLE-Anweisung in einer Übersetzungseinheit darf einen Namen haben.
3. Der Namenseintrag wird bei Ausgabe in die EAM-Datei (nicht bei Ausgabe in eine Bibliothek) als Identifikation in jeden Satz des Objektmoduls ausgegeben.

>>>>> siehe auch PRINT-, EJECT- und SPACE-Anweisung

UNSTK USING- oder PRINT-Status restaurieren

Funktion

Die UNSTK-Anweisung setzt den USING-Status, bzw. die PRINT-Parameter, die zuvor mit STACK gesichert wurden, wieder als aktuell ein.

Format

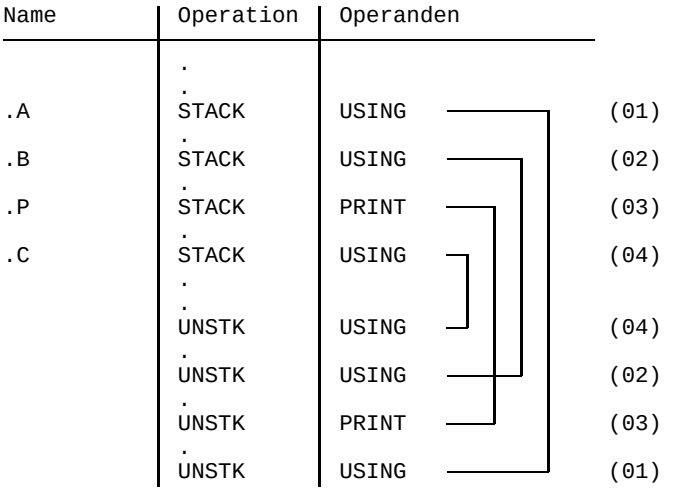
Name	Operation	Operanden
[.sym]	UNSTK	{PRINT[,...][,USING[,...]]} {USING[,...][,PRINT[,...]]}

.sym Folgesymbol

Programmierhinweis

Die UNSTK PRINT- bzw. UNSTK USING-Anweisung ruft jeweils diejenigen Werte ab, die mit der letzten vorhergehenden STACK PRINT- bzw. STACK USING-Anweisung gesichert wurden. Die nächste UNSTK-Anweisung ruft die Werte ab, die von der entsprechenden STACK-Anweisung zuvor gesichert wurden, usw.

Beispiel



(01) - (04) Die UNSTK-Anweisungen setzen jeweils den USING- bzw. PRINT-Status der STACK-Anweisung mit der gleichen Nummer als aktuell wieder ein.

>>>>> siehe auch STACK-Anweisung

USING Basisadreßregister zuweisen

Funktion

Die USING-Anweisung gibt eine Basisadresse an und weist dem Assembler ein oder mehrere Mehrzweckregister als Basisadreßregister zu.

Format 1

Name	Operation	Operanden
[.sym]	USING	adr,reg[[,reg][,...]]

.sym Folgesymbol
adr positiver absoluter oder relativer Ausdruck
reg Mehrzweckregister; positive absolute Ausdrücke, entweder
- Namen, denen ein absoluter Wert von 0 bis 15 zugewiesen wurde oder
- dezimale selbstdefinierende Werte von 0 bis 15

Beschreibung

adr gibt den Adreßwert an, den der Assembler für die Bildung von Distanzadressen benötigt.

reg gibt die Mehrzweckregister an, die als Basisadreßregister verwendet werden sollen.

Programmierhinweise

1. Die USING-Anweisung muß vor der ersten Verwendung von symbolischen Namen im Operanden einer Instruktion codiert werden. Sie wird vom Assembler zur Ablage der Adressen in Basis/Distanz-Form benötigt.
2. Die USING-Anweisung lädt keine Register.
   Damit zur Programmlaufzeit die Adreßzugriffe richtig ablaufen können, müssen die Basisregister mit den Basisadreßwerten geladen werden. Das erste wird mit dem BALR- bzw. BASR-Befehl geladen, die weiteren mit dem L- bzw. LM-Befehl (siehe Beispiel (02) und Assemblerbefehle, Beschreibung [3]). Der LA-Befehl ist hierfür ungeeignet.
3. Bei fehlender USING-Anweisung wird Register 0 als Basisregister verwendet.
4. Die Register 0 und 1 sollten nicht als Basisadreßregister benutzt werden, da sie von verschiedenen Assemblerbefehlen und Systemmakros verändert werden können.
5. Wenn für adr ein Stern (*) eingesetzt wird, muß die USING-Anweisung direkt nach dem BALR-Befehl folgen, damit BALR und USING dieselbe Adresse bezeichnen. Wenn für adr ein Name steht, muß dieser Name die Adresse des Befehls enthalten, bei dem die Distanzrechnung beginnen soll (siehe Beispiele (01) und (02)).
6. Wenn ein Basisadreßregister für die Adreßrechnung nicht ausreicht, müssen in der USING-Anweisung weitere Register als Basisadreßregister zugewiesen werden. Diese sollten mit den folgenden Basisadreßwerten geladen werden (siehe Beispiel (02)):

reg1	adr
reg2	adr+4096
reg3	adr+8192
.	.
.	.

7. Soll der Wert in einem augenblicklich benutzten Basisadreßregister geändert werden und die Distanzrechnung mit diesem neuen Wert fortgesetzt werden, so muß dieser neue Wert mit einer weiteren USING-Anweisung zugewiesen werden (siehe Beispiel (03)).
8. Wurden für einen Bereich mehrere Register als Basisadreßregister zugewiesen, dann verwendet der Assembler das Register, das die kleinste Distanzadresse liefert. Enthalten mehrere Register denselben Wert, dann wir das Register mit der höchsten Nummer verwendet.
9. Ist in einer vorangegangenen PRINT-Anweisung der Operand BASE gesetzt, wird nach jeder USING-Anweisung im Übersetzungsprotokoll der adressierbare Bereich ausgedruckt. Zusätzlich wird 19 Zeichen lang der Bemerkungseintrag der USING-Anweisung ausgegeben.

Beispiele

```
Name      Operation   Operanden                  
PROG      START
R0        EQU         0                        ┐
R1        EQU         1                        │ gelten für alle
R3        EQU         3                        } folgenden
R4        EQU         4                        │ Beispiele
R5        EQU         5                        ┘
          .
          .
          USING       BEG1,R3                  ┐
          BALR        R3,0                     } (01)
BEG1      EQU         *                        ┘
          .
          .
          BALR        R3,0                     ┐
          USING       *,R3,R4,R5               │
BEG2      EQU         *                        │
          LM          R4,R5,ACON               } (02)
          .                                    │
          .                                    │
ACON      DC          A(BEG2+4096,BEG2+2*4096) ┘
          .
          .
          USING       BEG2+1000,R3               (03)
          .
          DROP
          .
```

(01) Die USING-Anweisung enthält den Adreßpegel des Befehls, mit dem die Distanzrechnung beginnen soll.

(02) R3 soll den aktuellen Adreßpegel erhalten, daher ist die USING-Anweisung direkt nach dem BALR-Befehl codiert.
Die Register R4 und R5 werden durch den LM-Befehl mit den in ACON definierten Basisadreßwerten geladen.

(03) Ab dieser USING-Anweisung wird die Distanzrechnung mit dem Wert BEG2+1000 fortgesetzt.

Das folgende Beispiel berücksichtigt die Standard-Verknüpfungskonventionen und den Anschluß anderer Programmiersprachen (siehe ASSEMBH, Benutzerhandbuch [1]).

```
Name      | Operation | Operanden
----------+-----------+-------------------------
          |     .     |
          |     .     |
          | BALR      | R10,0
          | USING     | *,R10
          |     .     |
          |     .     |
          | LA        | R15,=A(UPRO2)
          | BALR      | R14,R15
          |     .     |
          | DROP      | R10
          |     .     |
UPRO2     | CSECT     |
          | USING     | UPRO2,R15              (01)
          |     .     |
          |     .     |
          | BR        | R14
          | DROP      | R15
          |     .     |
```

(01) Ein Laden des Registers R15 ist hier nicht nötig, da die Adresse vom Aufruf des Unterprogramms UPRO2 her vorhanden ist. Allerdings würde ein "BALR R15,0" vor allen Eventualitäten sichern.

>>>>> siehe auch DROP-Anweisung

Format 2

Name	Operation	Operanden
	USING	*PRV,reg

reg Mehrzweckregister; positiver absoluter Ausdruck, entweder
- ein Name, dem ein absoluter Wert von 0 bis 15 zugewiesen wurde oder
- ein dezimaler selbstdefinierender Wert von 0 bis 15.

Beschreibung

Format 2 der USING-Anweisung bewirkt, daß für alle Befehle, die Pseudoregister betreffen, das in reg angegebene Register als Basisadreßregister verwendet wird.

Programmierhinweis

Das in reg angegebene Register muß zusätzlich mit der Anfangsadresse des Speicherbereichs für den Pseudoregistervektor geladen werden.

Beispiel zu Format 2 siehe Anhang 11.4

>>>>> siehe auch CXD- und DXD-Anweisung

WXTRN Bedingte EXTRN-Adresse kennzeichnen

Funktion

Die WXTRN-Anweisung kennzeichnet eine symbolische Adresse, die in einer Übersetzungseinheit verwendet wird, jedoch in einer anderen definiert ist. Die WXTRN-Anweisung entspricht der EXTRN-Anweisung, wird jedoch vom Binder anders verarbeitet.

Format

Name	Operation	Operanden
[.sym]	WXTRN	name[,...]

.sym Folgesymbol
name Name

Beschreibung

Die WXTRN-Anweisung unterdrückt die Autolink-Funktion des Binders. Der Binder kann diese Externadresse nur befriedigen, wenn er den passenden Modul aufgrund eines anderen Steuermechanismus finden kann.
.

Weitere Regeln siehe EXTRN-Anweisung.

>>>>> siehe auch EXTRN- und ENTRY-Anweisung

XDSEC Externen Pseudoabschnitt definieren

Funktion

Die XDSEC-Anweisung definiert einen externen Pseudoabschnitt oder eine Referenz auf einen externen Pseudoabschnitt.

Format

Name	Operation	Operanden
name	XDSEC	[{D[EFINITION] / R[EFERENCE]}]

Beschreibung

name kennzeichnet den Namen eines externen Pseudoabschnitts.

Das Längenmerkmal des Namens ist 1.

DEFINITION oder D
muß bei der Definition eines externen Pseudoabschnitts angegeben werden.

Mit der XDSEC D-Anweisung wird Speicherplatz für den externen Pseudoabschnitt reserviert.

Für die Definition eines externen Pseudoabschnitts wird ein neuer Adreßpegel aufgebaut und der Anfangswert Null zugewiesen.

Die XDSEC D-Anweisung übergibt Externinformationen an den Binder, damit dieser die Zugriffe auf den externen Pseudoabschnitt aus anderen Programmen befriedigen kann.

REFERENCE oder R
kennzeichnet die Referenz eines externen Pseudoabschnitts, damit Zugriffe auf diesen Pseudoabschnitt möglich sind.

Durch die Anweisung XDSEC R wird der Adreßpegel auf Null gesetzt und bleibt für den gesamten Abschnitt auf Null.

Der tatsächliche Adreßwert eines Zugriffs auf einen externen Pseudoabschnitt wird vom Binder in den entsprechenden Befehl eingetragen.

Programmierhinweise

1. In externen Pseudoabschnitten dürfen keine EQU-Anweisungen eingesetzt werden.
2. Arithmetische Ausdrücke im Operanden einer EQU-Anweisung dürfen keine Differenz von XDSEC-Elementen sein.
3. Für eine XDSEC-Anweisung ohne Operandenangabe, bzw. mit fehlerhafter Operandenangabe, wird Typ R angenommen. Ist bereits ein gleichnamiger externer Pseudoabschnitt definiert, wird dessen Typ übernommen.
4. In einer Übersetzungseinheit kann eine XDSEC-Anweisung jeweils nur als Typ R oder als Typ D vorliegen, nie für beide Typen gleichzeitig.

Beispiel

Im folgenden Beispiel wird in PROG1 ein Eingabesatz initialisiert und seine Struktur definiert. PROG2 und PROG3 verarbeiten Teilinformationen aus dem Eingabesatz.

```
Name      | Operation | Operanden
----------|-----------|-------------------------
PROG1     | START     |
R0        | EQU       |  0                    ⎫
R1        | EQU       |  1                    ⎪
R2        | EQU       |  2                    ⎬ gelten für alle 3
R13       | EQU       | 13                    ⎪ Programme im Beispiel
R14       | EQU       | 14                    ⎪
R15       | EQU       | 15                    ⎭
          | .         |
          | .         |
          | BALR      | R2,0
          | USING     | *,R2
          | USING     | INXD,R13
          | LA        | R13,IN
          | L         | R15,VPROG2
          | BALR      | R14,R15
          | L         | R15,VPROG3
          | BALR      | R14,R15
          | TERM      |
          | DROP      | R2
*         |           |
IN        | DC        | CL30'ANTON     MUELLER'
VPROG2    | DC        | V(PROG2)
VPROG3    | DC        | V(PROG3)
*         |           |
INXD      | XDSEC     | D                     ⎫
VORNAME   | DS        | CL10                  ⎬ (01)
NAME      | DS        | CL20                  ⎭
          | END       |
          | .         |
          | .         |
          | .         |
PROG2     | CSECT     |
          | .         |
          | .         |
          | USING     | *,R15
          | USING     | INXD,R13
          | MVC       | OVN,VORNAME             (03)
          | BR        | R14
          | DROP      | R15,R13
```

```
*
OVN         DS          CL10
*                                                    ┐
INXD        XDSEC       R                            } (02)
VORNAME     DS          CL10                         ┘
            END
            .
            .
            .
PROG3       CSECT
            USING       *,R15
            USING       INXD,R13
            MVC         ONAME,NAME                     (03)
            BR          R14
            DROP
*
ONAME       DS          CL20
*                                                    ┐
INXD        XDSEC       R                            } (02)
NAME        DS          CL20                         ┘
            END
```

(01) Definition des externen Pseudoabschnitts.
(02) Referenz des externen Pseudoabschnitts.
(03) Zugriff auf den externen Pseudoabschnitt.

>>>>> siehe auch CSECT- und DSECT-Anweisung

5 Struktur der Makrosprache

Die Makrosprache kann als zusätzliche Sprache aufgefaßt werden, die ebenfalls vom Assembler bearbeitet wird. Die folgenden Kapitel beschreiben die Sprachelemente und Anweisungen, die in der Makrosprache zusätzlich zu denen der Assemblersprache eingesetzt werden können.

Die Makrosprache ermöglicht es, häufig gebrauchte Folgen von Instruktionen nur einmal in Form einer Makrodefinition zu schreiben und sie dann mit nur jeweils einer Instruktion, dem Makroaufruf, ins Programm einzufügen. Durch Steuerung über Parameter können die eingefügten Instruktionen bei jedem Makroaufruf variieren. In der Makrodefinition kann mit Hilfe von Bedingungsanweisungen das Einfügen von Instruktionsfolgen von bestimmten Bedingungen abhängig gemacht werden. Außerdem kann durch variable Parameter der Text der erzeugten Instruktionen und Kommentare veränderlich gestaltet werden.

5.1 Aufruf und Definition von Makros

Der Makroaufruf ist eine Instruktion im Text des Assembler-Quellprogramms. Das Resultat der Verarbeitung eines Makroaufrufs ist die Makrogenerierung. Sie besteht aus den Assembleranweisungen, Assemblerbefehlen, Makroanweisungen und Makroaufrufen, die zusammen die Funktion ausführen, die vom Makroaufruf erwartet wird.

Der Operationseintrag des Makroaufrufs bezeichnet den Namen des aufgerufenen Makros. Dieser Name steht in der Makrodefinition im Operationseintrag der Musteranweisung. Das Format der Operanden des Makroaufrufs muß dem Format der Operanden der Musteranweisung entsprechen (siehe 7.1, Makroaufruf und Musteranweisung).

Die Makrogenerierung wird vom Assembler mit Hilfe der Makrodefinition durchgeführt. Die Makrodefinition wird gemäß den Operandenangaben im Makroaufruf modifiziert. Makrodefinitionen können vom Benutzer geschrieben werden oder sie werden als Teil des Betriebssystems als Systemmakros dem Benutzer zur Verfügung gestellt.

Die aufgelösten Makros werden im Assemblerprotokoll nicht mehr auf Stelle 10 und 16 ausgerichtet. Wegen der 64 Zeichen langen Namens- und Opcodefelder wird die ersetzte Makroanweisung möglichst der Originalanweisung im Makro nachgebildet.

Die folgende Beschreibung der Makrosprache bezieht sich nur auf Makros, die vom Benutzer selbst geschrieben werden. Die Systemmakros und ihre Verwendung sind in den BS2000-Handbüchern Makroaufrufe an den Ablaufteil, Beschreibung [6] und DVS, Einführung und Kommandoschnittstelle [5] zu finden.

5.1.1 Ablage der Makrodefinition

Die Definition eines Makro kann entweder im Quellprogramm selbst vorliegen oder in einer Makrobibliothek abgelegt sein.

Hat eine Makrodefinition im Quellprogramm denselben Namen, wie eine Makrodefinition in einer Bibliothek, dann wird die Makrodefinition im Quellprogramm benutzt. Stehen Makrodefinitionen in mehreren Bibliotheken mit demselben Namen, so wird die zuerst gefundene Makrodefiniton benutzt (Suchhierarchie siehe ASSEMBH, Benutzerhandbuch [1]).

Makrodefinition im Quellprogramm

Eine Makrodefinition im Quellprogramm muß in jedem Fall vor dem ersten Aufruf dieses Makros spezifiziert und durchlaufen werden. Wird die Makrodefinition mit Hilfe von Bedingungsanweisungen übersprungen, dann wird sie auch nicht eingelesen. Der Makro gilt dann als nicht bekannt im Quellprogramm. Wird ein solcher Makro aufgerufen, so wird noch in den Makrobibliotheken nach der Definition gesucht und, falls diese dort ebenfalls nicht vorliegt, wird der Makroaufruf als Fehler (unbekannter Operationscode; siehe ASSEMBH, Benutzerhandbuch [1]) gemeldet.

Falls für verschiedene Makrodefinitionen im Quellprogramm derselbe Name verwendet wird, gilt die zuletzt eingelesene Makrodefinition bis zum Auftreten der nächsten Definition desselben Namens. Der Assembler gibt in diesem Fall eine Warnung aus.
Durch Bedingungsanweisungen übersprungene Makrodefinitionen werden auch hier nicht berücksichtigt.

Makrodefinition innerhalb einer Makrodefinition

Es ist auch möglich, daß eine Makrodefinition innerhalb einer anderen vorliegt (innere Makrodefinition). D.h., daß eine Makrodefinition abhängig von einer Makrogenerierung eingelesen wird (siehe auch 5.1.2, Aufbau der Makrodefinition).

Auch diese innere Makrodefinition muß vor ihrem ersten Aufruf durchlaufen werden. Erst danach ist sie auch im äußeren Makro bekannt.

Nachdem die innere Makrodefinition das erste Mal durchlaufen wurde, kann sie auch unabhängig von der äußeren aufgerufen werden.

Die äußere Makrodefinition kann im Quellprogramm liegen oder in einer Bibliothek abgelegt sein. Die innere Makrodefinition wird abgelegt, als wäre sie aus einer Bibliothek eingelesen worden.

Die Schachtelungstiefe für innere Makrodefinitionen ist, abhängig von der Speicherkapazität, beliebig.

Makrodefinition in einer Bibliothek

Eine Makrodefinition kann für viele Programme zugänglich gemacht werden, wenn sie in einer Bibliothek abgelegt wird. Damit der Assembler die Makrodefinition auffinden kann, muß ihm die entsprechende Bibliothek vor der Übersetzung zugewiesen werden (siehe ASSEMBH, Benutzerhandbuch [1]).

Wenn eine Makrodefinition in einer Bibliothek vorliegt, wird sie nur dann eingelesen, wenn ihr Aufruf durchlaufen wird. Wird der Aufruf übersprungen und nicht durchlaufen, dann wird die Makrodefinition auch nicht eingelesen.

5.1.2 Aufbau der Makrodefinition

Eine Makrodefinition besteht aus den folgenden vier Teilen in der aufgeführten Reihenfolge:

- **der Anfangsanweisung MACRO;**
  Sie kennzeichnet den Anfang einer Makrodefinition und muß jeweils die erste Anweisung in der Definition sein (siehe 7.2, MACRO-Anweisung).
- **der Musteranweisung;**
  Sie gibt den Namen des Makro an und die symbolischen Parameter, die in dieser Makrodefiniton vorkommen.
  Das Operandenformat der Musteranweisung bestimmt das Operandenformat des zugehörigen Makroaufrufs (siehe 7.1, Makroaufruf und Musteranweisung).
- **keiner, einer oder mehreren Modellanweisungen;**
  Sie ergeben nach der Übersetzung die gewünschte Folge von Instruktionen.
  Modellanweisungen können sein:
  - alle Assembleranweisungen, außer der ICTL-Anweisung,
  - alle Assemblerbefehle,
  - alle Makroaufrufe und
  - alle Makroanweisungen.
- **der Endanweisung MEND;**
  Sie kennzeichnet das Ende einer Makrodefinition und muß die jeweils letzte Anweisung in der Definition sein (siehe 7.2, MEND-Anweisung).

Zusätzlich zu den genannten Bestandteilen sind Kommentarzeilen möglich, die an jeder beliebigen Stelle innerhalb der Makrodefinition stehen können (siehe 5.2, Kommentare).

Beispiel

Das Beispiel zeigt in einfacher Form die mögliche Logik einer Makrodefinition.

```
Name        | Operation | Operanden
------------+-----------+-------------------------
            | MACRO     |                          (01)
&LABEL      | VARPAR    | &A,&B                    (02)
            | LCLC      | &C                       (03)
&LABEL      | MVC       | &A,&B                    (04)
            | .         |
            | .         |
&C          | SETC      | 'WORK'                   (05)
            | .         |
            | .         |
            | AIF       | ('&C' EQ 'WORK').ONE     (06)
            | AIF       | ('&C' EQ '').TWO         (07)
            | .         |
            | .         |
            | AGO       | .TWO                     (08)
            | .         |
            | .         |
.ONE        | ANOP      |                          (09)
            | .         |
            | .         |
.TWO        | ANOP      |                          (10)
            | .         |
            | .         |
            | MEND      |                          (11)
```

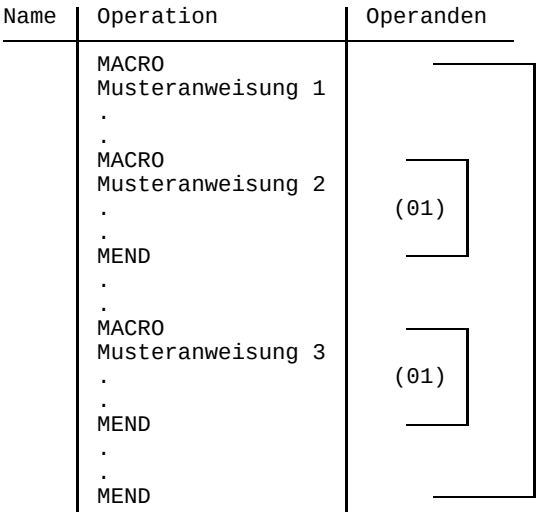
(01) Anfangsanweisung
(02) Musteranweisung mit dem Makronamen VARPAR und den symbolischen Parametern &LABEL, &A und &B
(03) bis (10) Modellanweisungen:
(03) Definition des lokalen SET-Parameters &C
(04) Der symbolische Parameter &LABEL repräsentiert über die Ersetzung in der Musteranweisung die symbolische Adresse des Makroaufrufs
(05) Wertzuweisung für &C
(06) und (07)
bedingte Verzweigungen, das Sprungziel ist abhängig vom Inhalt von &C
(08) unbedingte Verzweigung
(09) und (10)
Definitionen der Sprungziele .ONE und .TWO
(11) Endanweisung

5.1.3 Aufbau der inneren Makrodefinition

Eine Makrodefinition, die innerhalb einer anderen liegt, wird als innere Makrodefinition bezeichnet.

Diese innere Makrodefinition kann nicht mit variablen Parametern generiert werden. D.h., es wird keine textuelle Ersetzung (siehe 5.3.2, Variable Parameter) durchgeführt, während die Makrodefinition eingelesen wird. Über den äußeren Makro kann nur gesteuert werden, ob die innere Makrodefinition eingelesen werden soll oder nicht.

Eine Makrodefinition, die eine innere Makrodefinition enthält, kann nur nach dem folgenden Schema aufgebaut werden.



(01) Hier können weitere innere Makrodefinitionen geschachtelt sein.

5.2 Instruktionen und Kommentare

Der Text eines Assembler-Makro besteht, ebenso wie der eines Assembler-Quellprogramms aus einer Folge von Instruktionen und Kommentaren. Die zusätzlichen Möglichkeiten, die die Makrosprache für Instruktionen und Kommentare bietet, werden im folgenden beschrieben. Die Aussagen über die Struktur der Assemblersprache, die im Kapitel 2 gemacht wurden, gelten hier nach wie vor.

Instruktionen

Im einzelnen können Instruktionen der Makrosprache aus fünf Einträgen bestehen:
- dem Namenseintrag,
- dem Operationseintrag,
- dem Operandeneintrag,
- dem Bemerkungseintrag und
- dem Fortsetzungszeichen.

Diese Einträge müssen der obigen Reihenfolge entsprechen und, bis auf das Fortsetzungszeichen, durch mindestens ein Leerzeichen voneinander getrennt sein.

Namens-, Operations- und Operandeneintrag verschiedener Instruktionen, auch von Assembleranweisungen und Assemblerbefehlen, können bei entsprechenden Voraussetzungen mit Hilfe von **variablen Parametern** generiert werden.

Für eine Instruktion der Makrosprache sind neun Fortsetzungszeilen erlaubt. Eine Sonderstellung haben hier der Makroaufruf, die Musteranweisung, die Anweisungen GBLx und LCLx, sowie Format 2 von AIF und AGO; für sie gibt es eine alternative Formatdarstellung (siehe 7.1.4, Alternatives Anweisungsformat)

Kommentare

Kommentare können innerhalb der Makrodefiniton an beliebiger Stelle auftreten. In der Makrosprache gibt es, wie in der Assemblersprache, zwei Möglichkeiten, Kommentare zu kennzeichnen:

* Mit einem Stern in der Anfangsspalte;
Diese Kommentarzeilen werden bei der Makroauflösung vom Assembler übernommen und im Übersetzungsprotokoll ausgedruckt.

.* Mit einem Punkt in der Anfangsspalte, dem ein Stern folgt;
Diese Kommentarzeilen dienen nur zur Dokumentation der Makrodefinition. Sie werden vom Assembler bei der Makroauflösung nur bei Makrodefinitionen im Quellprogramm übernommen und ausgedruckt, bei Makrodefinitionen in Bibliotheken nicht.

Wie im Bemerkungseintrag von Instruktionen werden variable Parameter in Kommentarzeilen nicht ersetzt. Eine Generierung von variablen Kommentarzeilen ist nur über die Anweisung MNOTE *,... möglich (siehe 7.2)

Ausschalten der Funktion von Instruktionen

Sind im Namenseintrag von Instruktionen symbolische Parameter, bzw. SETC-Parameter zugelassen, kann man aus Instruktionen in der Makrodefinition bzw. im Quellprogramm Kommentare erzeugen, d.h., man kann ihre Funktion ausschalten.

In diesem Fall wird dem symbolischen Parameter oder SETC-Parameter für die Makrogenerierung ein * zugewiesen (.* ist nicht zulässig). Dadurch wird die Instruktion als Kommentar interpretiert.

5.3 Namenseintrag

Im Namenseintrag steht eine Folge von maximal 64 Buchstaben und Ziffern, die dazu dient, eine Instruktion zu identifizieren.

Der Namenseintrag kann wahlfrei sein. Wenn er vorhanden ist, muß er in der Anfangsspalte beginnen. Ist die Anfangsspalte leer, so nimmt der Assembler an, daß kein Namenseintrag vorhanden ist und interpretiert die folgenden Zeichen als Operationscode.

Im Namenseintrag einer Instruktion kann stehen:
- ein Name (siehe 2.3),
- ein Folgesymbol,
- ein variabler Parameter oder
- eine Verkettung von variablen Parametern und alphanumerischen Zeichen.

5.3.1 Folgesymbole

Ein Folgesymbol im Namenseintrag ermöglicht es, sich in den Operanden der Bedingungsanweisungen AIF und AGO auf diese Instruktion zu beziehen, d.h., sie als Sprungziel zu kennzeichnen. Dadurch wird es möglich, die Reihenfolge zu variieren, in der die Instruktionen verarbeitet werden.

Folgesymbole sind lokale Symbole, d.h., sie sind nur innerhalb der jeweiligen Makrodefinition bekannt, in der sie definiert wurden. Wird dasselbe Folgesymbol innerhalb und außerhalb einer Makrodefinition verwendet oder in zwei verschiedenen Makrodefinitionen, dann gilt es in jedem Fall als eigenes Symbol.

Ein Folgesymbol im Namenseintrag einer generierten Instruktion wird im Übersetzungsprotokoll nicht ausgedruckt.

Regeln

- Folgesymbole dürfen maximal aus 65 Zeichen bestehen.
- Das erste Zeichen eines Folgesymbols muß ein Punkt (.) sein, das zweite Zeichen ein Buchstabe. Diesem können bis zu 63 weitere Buchstaben und/oder Ziffern folgen.
- Im Namenseintrag von Instruktionen dürfen Folgesymbole nur in diesem Standardformat geschrieben werden.
- Folgesymbole im Operandeneintrag können zusätzlich im generierten Format geschrieben werden. Folgesymbole im generierten Format bestehen aus einem Punkt, dem ein variabler Parameter folgt oder eine Verkettung von variablen Parametern und alphanumerischen Zeichen (siehe 5.5, Operandeneintrag).

– Leerzeichen innerhalb eines Folgesymbols sind nicht zulässig.
– Pro Makro sind maximal 2^{15}-1 Folgesymbole erlaubt.

Beispiele für zulässige Folgesymbole

```
.LOOP                   .A123456789
.LOOP_2                 .A@B4
.Loop                   .$ABC
                        .ABC&PARAM
```

Beispiele für unzulässige Folgesymbole

```
AREA                    (erstes Zeichen kein Punkt)
.1ABC                   (zweites Zeichen kein Buchstabe)
.LOOP*                  (enthält das Sonderzeichen *)
.LOOP␣1                 (enthält ein Leerzeichen)
```

5.3.2 Variable Parameter

Variable Parameter werden für textuelle Ersetzungen im Namens-, Operations- und Operandeneintrag einer Instruktion verwendet. Über Makroanweisungen oder direkt durch den Assembler können ihnen Werte zugewiesen werden. Die textuelle Ersetzung erfolgt dann mit dem augenblicklich aktuellen Wert. Textuelle Ersetzungen im Bemerkungseintrag erfolgen nicht. Bei Verwendung in Bedingungsanweisungen können variable Parameter zur Steuerung der Assemblierung verwendet werden.

Bei welchen Instruktionen der Assembler-, bzw. Makrosprache variable Parameter im Namenseintrag erlaubt sind, ist aus den jeweiligen Formatbeschreibungen ersichtlich. Bei den Assemblerbefehlen sind variable Parameter im Namenseintrag grundsätzlich möglich.

Pro Makrodefinition sind maximal 2^{15}-1 variable Parameter erlaubt.

Regeln

- Ein variabler Parameter darf aus maximal 64 Zeichen bestehen.
- Das erste Zeichen muß ein kaufmännisches Und (&) sein.
- Leerzeichen innerhalb eines variablen Parameters sind nicht zulässig.

Die Zeichen, die bei der textuellen Ersetzung einen variablen Parameter ersetzen, müssen mit den Syntaxregeln des Eintrags übereinstimmen, in dem der variable Parameter auftritt.

Beispiel

`&PAR` im Namenseintrag kann nicht ersetzt werden durch `NAME␣BALR`

Innerhalb des Operandeneintrags, außer beim Makroaufruf, ist es dagegen möglich, aus einem variablen Parameter zwei oder mehrere Operanden zu generieren.

Beispiel

`&PAR` im Operandeneintrag kann ersetzt werden durch `FIELD1,FIELD2`

Zu den variablen Parametern gehören:

- **symbolische Parameter;**
  Sie werden in der Musteranweisung definiert und der Programmierer weist ihnen bei jedem Makroaufruf neue Werte zu.
- **variable Systemparameter;**
  Sie beginnen mit der Zeichenfolge &SYS... und haben festgelegte Bedeutungen. Der Assembler weist ihnen einen Wert zu, wenn er eine Makrodefinition verarbeitet oder bei Beginn einer Übersetzung.
- **SET-Parameter;**
  Sie sind variable Hilfsfelder, denen der Programmierer innerhalb oder außerhalb einer Makrodefinition über Makroanweisungen einen Wert zuweisen kann.

Symbolische Parameter und ein Teil der variablen Systemparameter sind lokale Symbole. D.h., ihre Werte werden jeweils beim Beginn der Verarbeitung eines Makros zurückgesetzt. Sie sind also nur innerhalb eines Makros bekannt.

SET-Parameter können lokale oder globale Symbole sein. Wenn sie als global definiert wurden, kann man sie zur Weitergabe von Werten zwischen Makros oder zwischen Makro und Assembler-Quellprogramm verwenden.

Hinweis

Eine genaue Erörterung der variablen Parameter und ihrer Verwendung erfolgt in Kapitel 6, Variable Parameter.

5.3.3 Generierte variable Parameter

Es ist generell möglich, den Namen eines variablen Parameters mit Hilfe eines oder mehrerer weiterer variabler Parameter zu generieren.

Dabei müssen die Zeichen, die das Ergebnis der textuellen Ersetzung darstellen, den Syntaxregeln des Eintrags entsprechen, in dem der generierte variable Parameter verwendet wird.

Format von generierten variablen Parametern:

&(ele)[(d)]

ele kann sein:
- variabler Parameter: &par
- indizierter SET-Parameter (siehe 6.2): &par(d)
- generierter variabler Parameter: &(ele)
- generierter indizierter SET-Parameter: &(ele)(d)
- alphanumerische Zeichen: val
- Verkettung der o.g. Grundelemente (siehe 5.3.4)

d kann sein:
- Index, arithmetischer Makroausdruck: arexp
- Operandenunterliste mit arithmetischen Makroausdrücken (siehe 7.1.2): arexp,arexp[,...]

Beispiele

```
&(&PAR1.&(&PAR2(D1)).AB)  ergibt  &A1AB   wenn gilt   &PAR1      SETC  'A'
                                                     &PAR2(D1)  SETC  'B'
                                                     &B         SETA  1

&(&(&(AB)(D1))(D2))       ergibt  &PAR2   wenn gilt   &AB(D1)    SETC  'PAR1'
                                                     &PAR1(D2)  SETC  'PAR2'
```

5.3.4 Verkettung von variablen Parametern und alphanumerischen Zeichen

Variable Parameter können miteinander oder mit alpanumerischen Zeichen verkettet werden. Die Zeichenfolge, die sich auf Grund der Verkettung im generierten Text ergibt, muß mit den Syntaxregeln des Eintrags übereinstimmen, in dem die Verkettung vorkommt.

– Sollen alphanumerische Zeichen auf einen variablen Parameter folgen, müssen sie durch einen Punkt getrennt werden. Soll der Punkt im generierten Text erscheinen, müssen in der Makrodefinition zwei Punkte geschrieben werden.

  Beispiel (Der aktuelle Wert von `&PARAM` ist `NAME`)

`&PARAM.ABC`	ergibt	`NAMEABC`
`&PARAM..ABC`	ergibt	`NAME.ABC`

– Soll ein variabler Parameter auf alphanumerische Zeichen folgen, dann ist kein Punkt erlaubt. Soll im generierten Text ein Punkt erscheinen, darf in der Makrodefinition nur ein Punkt geschrieben werden.

  Beispiel

`ABC&PARAM`	ergibt	`ABCNAME`
`ABC.&PARAM`	ergibt	`ABC.NAME`

– Sollen variable Parameter miteinander verkettet werden, können sie durch Aneinanderreihen oder durch einen Punkt getrennt verkettet werden. Soll der Punkt im generierten Text erscheinen, müssen auch hier in der Makrodefinition zwei Punkte geschrieben werden.

  Beispiel (Der aktuelle Wert von `&PARAM1` ist `NAME1` und von `&PARAM2` `NAME2`)

`&PARAM1&PARAM2`	ergibt	`NAME1NAME2`
`&PARAM1.&PARAM2`	ergibt	`NAME1NAME2`
`&PARAM1..&PARAM2`	ergibt	`NAME1.NAME2`

  Im Namenseintrag darf eine Verkettung von variablen Parametern und alphanumerischen Zeichen nur bei Assembleranweisungen und Assemblerbefehlen verwendet werden, nicht bei Makroanweisungen. Für Operations- und Operandeneintrag ist die Verwendung einer solchen Verkettung abhängig vom Format der Instruktion.

5.4 Operationseintrag

Der Operationseintrag kann enthalten:
- Den mnemotechnischen Operationscode einer Makroanweisung, einer Assembleranweisung, oder eines Assemblerbefehls oder
- den Namen eines System- oder Benutzermakros (Makroaufruf) oder
- einen variablen Parameter oder
- eine Verkettung von variablen Parametern und alphanumerischen Zeichen.

Ein Operationseintrag muß angegeben werden. Er muß durch mindestens ein Leerzeichen vom Namenseintrag getrennt sein, bzw. bei fehlendem Namenseintrag, mindestens eine Stelle rechts von der Anfangsspalte beginnen.

Ein zulässiger Operationseintrag besteht aus dem mnemotechnischen Operationscode einer Assembleranweisung, eines Assemblerbefehls oder einer Makroanweisung. Namen von Benutzermakros müssen nach den Regeln von Namen (siehe 2.3, Namenseintrag) gebildet werden.

Variable Parameter im Operationseintrag

Der Operationseintrag kann mit Hilfe von variablen Parametern oder durch Verkettung von variablen Parametern und alphanumerischen Zeichen generiert werden. Die Zeichen, die einen variablen Parameter ersetzen, müssen mit den Syntaxregeln des Operationseintrags übereinstimmen.

Folgende Instruktionen dürfen **nicht** mit variablen Parametern generiert werden:
- alle Makroanweisungen,
- die COPY-Anweisung,
- die ICTL-Anweisung,
- die MNOTE-Anweisung und
- die REPRO-Anweisung.

Alle anderen Instruktionen der Assemblersprache, sowie Namen von System- und Benutzermakros können mit Hilfe von variablen Parametern generiert werden.

Für variable Parameter im Operationseintrag gelten die gleichen Aussagen, die bereits für variable Parameter im Namenseintrag gemacht wurden (siehe 5.2.1.2 bzw. 6, Variable Parameter).

Beispiel

```
Name          | Operation | Operanden
--------------+-----------+-----------------
* Makrodefinition
              | MACRO     |
              | EXOP      | &OP,&BASREG
              | .         |
              | .         |
              | &OP       | &BASREG,0
              | USING     | *,&BASREG
              | .         |
              | .         |
              | MEND      |

* Makroaufruf 1
              | EXOP      | BASR,2

* Generierte Instruktionen
              | BASR      | 2,0
              | USING     | *,2

* Makroaufruf 2
              | EXOP      | BALR,2

* Generierte Instruktionen
              | BALR      | 2,0
              | USING     | *,2
```

5.5 Operandeneintrag

Der Operandeneintrag von Instruktionen in der Makrosprache kann aus einem oder mehreren Operanden bestehen, die ihrerseits einen oder mehrere Ausdrücke enthalten können.

Die Operanden müssen durch Kommas getrennt sein. Zwischen den Operanden und den Trennkommas dürfen, außer beim alternativen Anweisungsformat (siehe 7.1.4), keine Leerzeichen stehen.

Der Operandeneintrag kann wahlfrei sein. Wenn er angegeben ist, muß er durch mindestens ein Leerzeichen vom Operationseintrag getrennt sein.

5.5.1 Variable Parameter im Operandeneintrag

Variable Parameter können im Operandeneintrag von Assemblerinstruktionen und von Makroinstruktionen eingesetzt werden. Bei den Assemblerinstruktionen werden sie zur textuellen Ersetzung verwendet. Bei den Makroinstruktionen ist aus der jeweiligen Formatbeschreibung ersichtlich, ob ein variabler Parameter möglich ist und wie er ausgewertet wird.

Für variable Parameter im Operandeneintrag gelten ansonsten die gleichen Aussagen, die bereits für variable Parameter im Nameneintrag gemacht wurden (siehe 5.3.2, bzw. Kapitel 6, Variable Parameter).

Beispiel

```
Name        | Operation | Operanden
------------------------------------
* Makrodefinition
            | MACRO     |
            | EX1       | &PAR1,&PAR2
            | MVC       | &PAR1,&PAR2
            | .         |
            | .         |
            | MEND      |

* Makroaufruf
            | EX1       | FIELD1,FIELD2

* Generierte Instruktionen
            | MVC       | FIELD1,FIELD2
            | .         |
            | .         |
FIELD1      | DS        | CL4
FIELD2      | DC        | C'ABCD'
```

5.5.2 Folgesymbole im Operandeneintrag

Zusätzlich zu den Makroausdrücken kann der Operandeneintrag der AIF- und AGO-Anweisung Folgesymbole enthalten. Die Folgesymbole im Operandeneintrag geben die Instruktionen an, zu denen nach der Verarbeitung der AIF- und AGO-Anweisung verzweigt werden soll.

Folgesymbole können im Operandeneintrag im Standardformat (siehe 5.3.1) oder im generierten Format geschrieben werden.

Ein Folgesymbol im Standardformat beginnt mit einem Punkt (.), das zweite Zeichen muß ein Buchstabe sein, und diesem können noch 63 weitere Buchstaben und Ziffern folgen.

Ein Folgesymbol im generierten Format beginnt ebenfalls mit einem Punkt. Diesem folgt ein variabler Parameter oder ein Name, der mit einem variablen Parameter verkettet ist.

Beispiele für Folgesymbole im generierten Format

```
.&LOOP
.LOOP&NAME
.&Loop
.NAME123&LOOP
.&LOOP.NAME
```

5.5.3 Makroausdrücke

Ausdrücke sind auch in der Makrosprache die Grundbestandteile der Operanden von Instruktionen. Sie sind aus Elementen und Operatoren aufgebaut.
In der Makrosprache gibt es arithmetische Makroausdrücke, Zeichenausdrücke, Vergleichsausdrücke und logische Ausdrücke.

Die Art des Ausdrucks und damit seine Berechnung wird bestimmt durch die Operatoren. Eine Zusammenfassung dieses Sachverhalts für die verschiedenen Ausdrücke und die jeweils möglichen Elemente der Ausdrücke sind in Bild 5-1 zu finden.

Arithmetische Makroausdrücke und logische Ausdrücke können aus jeweils nur einem Element ohne Operatoren bestehen. Der Assembler interpretiert solche einfachen Ausdrücke abhängig von ihrer jeweiligen Verwendung als arithmetisch oder logisch.

Zeichenausdrücke bestehen aus nur einem Element, sie enthalten keine Operatoren.

Vergleichsausdrücke dagegen müssen aus zwei Elementen und einem Operator aufgebaut sein.

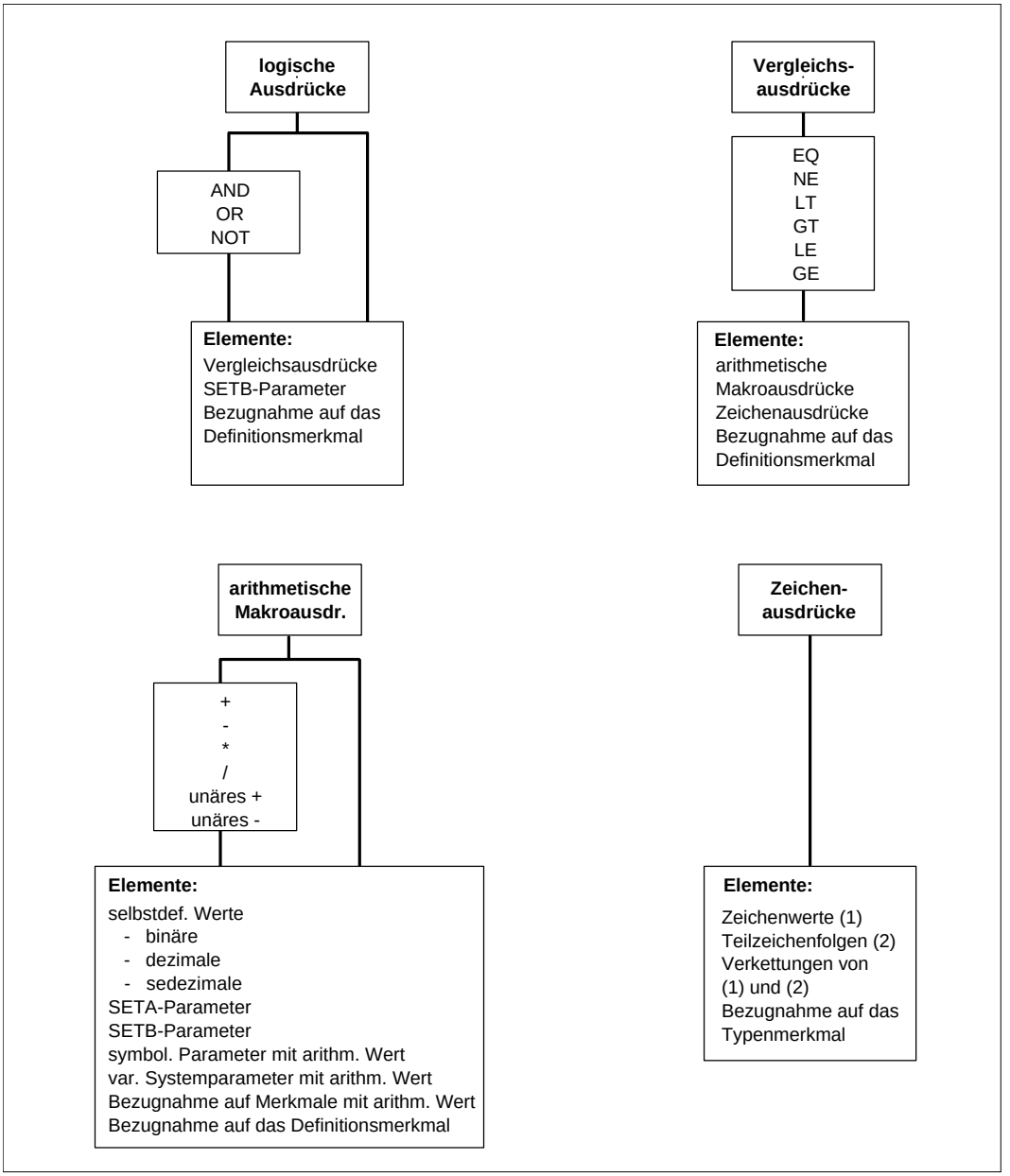


Bild 5-1 Makroausdrücke

5.5.4 Zeichenausdrücke

Ein Zeichenausdruck kann sein:
- ein Zeichenwert,
- eine Teilzeichenfolge,
- eine Verkettung von Zeichenwerten und Teilzeichenfolgen oder
- eine Bezugnahme auf das Typenmerkmal (siehe 5.5.8.1).

Ein Zeichenausdruck enthält keine Operatoren.

5.5.4.1 Zeichenwert

Ein Zeichenwert besteht aus einer beliebigen Kombination von Zeichen, die in Hochkommas eingeschlossen ist.

```
Format von Zeichenwerten        'val'
```

val — Zeichenfolge oder
variabler Parameter oder
Verkettung von variablen Parametern und alphanumerischen Zeichen

Ein variabler Parameter innerhalb eines Zeichenwertes wird durch seinen aktuellen Wert ersetzt.

Wird ein variabler Parameter, der einen Zeichenwert darstellt, als Element in einem Vergleichsausdruck verwendet, dann können die beiden Hochkommas wegfallen.

Wird ein SETA- oder SETB-Parameter verwendet, dann ist das Ergebnis die Zeichendarstellung des dezimalen oder booleschen Wertes, ohne Vorzeichen (Absolutbetrag) und ohne führende Nullen.

Ein Zeichenwert darf aus maximal 1020 Zeichen bestehen.
Ebenso darf die Auflösung eines variablen Parameters nur eine Zeichenfolge von maximal 1020 Zeichen ergeben.

Hochkomma in Zeichenwerten

Soll die Auflösung eines Zeichenwertes nach der Generierung ein Hochkomma enthalten, dann müssen im Zeichenwert selbst zwei Hochkommas geschrieben werden.

Beispiele

```
'NUM''ONE'          ergibt      NUM'ONE
'L''SYMBOL'         ergibt      L'SYMBOL
''''                ergibt      '
```

&-Zeichen in Zeichenwerten

Ein &-Zeichen in einem Zeichenwert wird vom Assembler als Beginn eines variablen Parameters interpretiert. Anders als bei der Auflösung von zwei Hochkommas werden zwei &-Zeichen in einem Zeichenwert dagegen auch als zwei &-Zeichen abgelegt. Stehen also in einem Zeichenwert zwei &-Zeichen hintereinander, dann werden sie auch bei der textuellen Ersetzung durch diesen Zeichenwert abgelegt.

Nur ein &-Zeichen in der Auflösung eines Zeichenwertes kann nur mit Hilfe einer Teilzeichenfolge erreicht werden (siehe unten).

Beispiel

```
Name       | Operation | Operanden
-----------+-----------+-------------------------
&FIRMA     | SETC      | 'NAME && CO'          (01)
           | DC        | C'&FIRMA'             (02)

* generierte Instruktion
           | DC        | C'NAME && CO'         (03)

*erzeugte Konstante  NAME & CO
```

(01) Dem variablen Parameter &FIRMA wird der Wert NAME && CO zugewiesen.
(02) Die C-Konstante wird mit dem variablen Parameter definiert.
(03) Nach der Makroauflösung hat die C-Konstante den Inhalt NAME && CO. Dieser Wert entspricht der Assembler-Syntax und wird als NAME & CO übersetzt.

5.5.4.2 Teilzeichenfolge

Teilzeichenfolgen ermöglichen es, einen Ausschnitt aus einem Zeichenwert anzusprechen.

```
Format von Teilzeichenfolgen          [(dup)]'val'(a,b)
```

dup — Wiederholungsfaktor; arithmetischer Makroausdruck

Der Wiederholungsfaktor ist nur für Teilzeichenfolgen im Operandeneintrag von SETC-Anweisungen zugelassen. Teilzeichenfolgen in Vergleichsausdrücken dürfen **keinen** Wiederholungsfaktor haben.

val — Zeichenwert

a — Nummer des Zeichens, bei dem der Ausschnitt aus dem Zeichenwert beginnen soll;
dezimaler, selbstdefinierender Wert oder arithmetischer Makroausdruck

b — Zahl der Zeichen, die der Ausschnitt aus dem Zeichenwert enthalten soll;
dezimaler selbstdefinierender Wert oder arithmetischer Makroausdruck

Bei der Auflösung einer Teilzeichenfolge wird zuerst der Ausschnitt aus dem Zeichenwert hergestellt und anschließend der Wiederholungsfaktor ausgewertet.

Eine Teilzeichenfolge darf, ebenso wie ihre Auflösung, nach der Auswertung des Wiederholungsfaktors nur aus maximal 1020 Zeichen bestehen.

Wenn in einer Teilzeichenfolge der Wert a größer ist als die Zahl der Zeichen im Zeichenwert, erhält man als Ergebnis eine leere Zeichenfolge. Ist der Wert b größer als die Zahl der Zeichen, dann werden nur die vorhandenen Zeichen im Zeichenwert als Ergebnis eingesetzt (siehe Beispiele).

Beispiele

```
(3)'TEXT'(2,2)    ergibt     EXEXEX
'VAL &&'(1,5)     ergibt     VAL &
(3)'&PAR'(1,&A)   ergibt     ABABAB  ⎫ wenn &PAR den aktuellen Wert ABC
'&PAR.%4'(1,4)    ergibt     ABC%    ⎭ und &A den aktuellen Wert 2 hat.
'ABCD'(12,4)      ergibt     leere Zeichenfolge
'ABCD'(1,10)      ergibt     ABCD
```

5.5.4.3 Verkettung von Zeichenwerten und Teilzeichenfolgen

Zeichenwerte und Teilzeichenfolgen können in beliebiger Reihenfolge miteinander verkettet werden:

Für alle Verkettungen gilt:

– Zwischen Hochkomma und Hochkomma **muß** ein Punkt stehen.

Beispiele

`'ABC'.'DEF'`	ergibt	`ABCDEF`
`'ABC'.'DEF'(2,1)`	ergibt	`ABCE`

– Zwischen Hochkomma und linker Klammer **muß** ein Punkt stehen (Wiederholungsfaktor).

Beispiele

`'ABC'.(2)'XY'`	ergibt	`ABCXYXY`
`'BIN'.(2)'XY'(1,1)`	ergibt	`BINXX`

– Zwischen rechter Klammer und Hochkomma **kann** ein Punkt stehen (Teilzeichenfolge).

Beispiele

`'PAR123'(1,3).'AB'`	ergibt	`PARAB`
`'PAR123'(1,3)'AB'`	ergibt	`PARAB`

– Zwischen rechter und linker Klammer **kann** ein Punkt stehen.

Beispiele

`'PAR123'(1,3).(2)'AB'`	ergibt	`PARABAB`
`'PAR123'(1,3)(2)'AB'`	ergibt	`PARABAB`

Hinweis

Auch hier gilt, daß Teilzeichenfolgen in Vergleichsausdrücken keinen Wiederholungsfaktor haben dürfen.

5.5.5 Arithmetische Makroausdrücke

Ein arithmetischer Makroausdruck setzt sich zusammen aus Elementen und arithmetischen Operatoren (siehe 2.5.1) oder er besteht aus nur einem Element ohne Operatoren.

Folgende Elemente sind in einem arithmetischen Makroausdruck erlaubt:

- binäre, dezimale und sedezimale selbstdefinierende Werte (siehe 2.5.2),
- SETA-Parameter (siehe 6.2),
- SETB-Parameter (siehe 6.2),
- SETC-Parameter, symbolische Parameter und variable Systemparameter, sofern sie einen dezimalen, binären oder sedezimalen Wert haben (siehe 6.1 bis 6.3) und
- Bezugnahmen auf Merkmale mit einem arithmetischen Wert (Längenmerkmal, Zählermerkmal, Anzahlmerkmal, Skalenfaktormerkmal und Ganzzahligkeitsmerkmal, siehe 5.5.8),
- Bezugnahmen auf das Definitionsmerkmal (siehe 5.5.8.7).

Die aufgeführten Elemente sind in den jeweiligen Abschnitten dieses Handbuchs beschrieben.

In einem arithmetischen Makroausdruck kann man die Reihenfolge der Bearbeitung durch Klammern verändern. Eine Schachtelung der Klammern ist möglich.

Regeln

- Ein arithmetischer Makroausdruck darf nicht mit einem Operator, außer dem unären Plus und dem unären Minus beginnen.
- In einem arithmetischen Makroausdruck dürfen zwei Elemente nicht unmittelbar aufeinander folgen.
- Das unäre Plus und das unäre Minus dürfen allen anderen Operatoren direkt folgen.
- Die Endwerte der Berechnung von arithmetischen Makroausdrücken müssen zwischen -2^{31} und $+2^{31}-1$ liegen.

Beispiele für zulässige arithmetische Makroausdrücke

```
&AREA+X'2D'
&EXIT-S'&ENTRY+1
&AREA+X'2D'/(&EXIT-S'&ENTRY+1)
I'&N/25
50
```

Berechnung von arithmetischen Makroausdrücken

1. Jedem Element wird sein numerischer Wert zugeordnet.
2. Arithmetische Operationen werden von links nach rechts ausgeführt. Multiplikation und Division kommen vor Addition und Subtraktion.
3. Bei Ausdrücken, die Klammern enthalten, werden zuerst die Werte in den Klammern errechnet. Bei mehrstufigen Klammern wird zuerst der innere Klammerausdruck berechnet.
4. Division durch Null ist zugelassen und liefert das Ergebnis Null.

Hinweis

Haben SETC-Parameter, symbolische Parameter und variable Systemparameter keinen arithmetischen Wert, kommt es zu einem Flag (nicht im F-ASSEMB-COMPATIBLE Modus) und der Ersatzwert 0 wird verwendet.

5.5.6 Vergleichsausdrücke

Ein Vergleichsausdruck setzt sich zusammen aus zwei Elementen und einem Vergleichsoperator. Das Ergebnis eines Vergleichsausdrucks kann 0 oder 1 (falsch oder wahr) sein.

Die folgenden Vergleichsoperatoren sind erlaubt:

EQ	gleich
NE	ungleich
LT	kleiner als
GT	größer als
LE	kleiner gleich
GE	größer gleich

Die Elemente des Vergleichsausdrucks bestimmen, ob es sich um einen arithmetischen Vergleich oder einen Zeichenvergleich handelt.

Bei einem **arithmetischen Vergleich** muß mindestens ein Element ein arithmetischer Makroausdruck sein.

Bei einem **Zeichenvergleich** sind beide Elemente Zeichenausdrücke.

Regeln

- Die Vergleichsoperatoren müssen von den Elementen durch jeweils ein Leerzeichen getrennt sein.
- Ein Vergleichsausdruck kann in Klammern gesetzt werden.
- Werden in einem Vergleichsausdruck variable Parameter als Zeichenwerte verwendet, können die Hochkommas wegfallen, die in allen anderen Fällen einen Zeichenwert kennzeichnen.
- In einem Zeichenvergleich können Elemente bis zu einer Länge von 1020 Zeichen verglichen werden. Sind in einem Zeichenvergleich die Elemente ungleich lang, dann wird stets das kürzere Element als das kleinere betrachtet.

Beispiele für zulässige Vergleichsausdrücke

&CHAR1 und &CHAR2 sollen SETC-Parameter sein und &AR1 und &AR2 SETA-Parameter. &PAR1 sei ein symbolischer Parameter, dem eine Zeichenfolge zugewiesen wurde und &PAR2 ein symbolischer Parameter, dem ein arithmetischer Wert zugewiesen wurde.

`'FIELD' NE '&CHAR1'`	Zeichenvergleich
`'FIELD' NE &CHAR1`	Zeichenvergleich
`&PAR1 EQ &CHAR1`	Zeichenvergleich
`&AR1 GT &AR2`	arithmetischer Vergleich
`&AR1 GT 16*&AR2+4`	arithmetischer Vergleich
`&AR1 GT 20`	arithmetischer Vergleich
`&PAR2 EQ &CHAR1`	Zeichenvergleich
`&PAR2 EQ &AR1`	arithmetischer Vergleich

5.5.7 Logische Ausdrücke

Logische Ausdrücke setzen sich zusammen aus Elementen und logischen Operatoren oder sie bestehen aus nur einem Element ohne Operatoren. Sie können als Ergebnis nur die logischen Werte 0 (falsch) oder 1 (wahr) erhalten.

Elemente eines logischen Ausdrucks können sein:

- SETB-Parameter (siehe 6.2),
- Vergleichsausdrücke (siehe 5.5.6) und
- Bezugnahmen auf das Definitonsmerkmal (siehe 5.5.8.7).

Die folgenden logischen Operatoren sind erlaubt:

AND	logisches Und
OR	inclusives Oder
NOT	Verneinung

In einem logischen Ausdruck kann man die Reihenfolge der Bearbeitung durch Klammern beeinflussen. Hierbei ist eine Schachtelung der Klammern möglich.

Regeln

- Die logischen Operatoren müssen jeweils durch ein Leerzeichen von den Elementen getrennt sein.
- Ein logischer Ausdruck darf nicht mit AND oder OR beginnen.
- In einem logischen Ausdruck dürfen nicht zwei Elemente direkt aufeinander folgen.
- Jedes Element eines logischen Ausdrucks kann in Klammern gesetzt werden. In diesem Fall ist zwischen Operator und eingeklammertem Element kein Leerzeichen notwendig.
- Die Operatoren AND und OR dürfen nicht miteinander kombiniert werden.
  Die Kombination mit NOT ist nur als AND NOT, bzw. OR NOT möglich.

Beispiele für zulässige logische Ausdrücke

```
&PAR1 AND &PAR2
&PARA OR &PARB
NOT &B AND &C
NOT(&BIN1 AND &BIN2)
(&BINA AND NOT &BINB)OR(&BINB AND NOT &BINA)
```

Die folgenden Beispiele zeigen zulässige logische Ausdrücke, in denen Zeichenvergleiche und arithmetische Vergleiche als Elemente verwendet werden.

```
&AREA+2 GT 29 OR &AR1
(&AREA+2 GT 20)OR(&AR1)
NOT &AR1 AND &AREA+2 GT 20
NOT &AR1 AND(&AREA+2 GT 20)
```

Berechnung von logischen Ausdrücken

Logische Ausdrücke werden nach folgenden Regeln auf einen Einzelwert zurückgeführt:

1. Jedes Element des logischen Ausdrucks wird berechnet und erhält seinen logischen Wert 0 oder 1 (falsch oder wahr).
2. Logische Operationen werden von links nach rechts ausgeführt. NOT wird jedoch vor AND und AND wird vor OR bearbeitet.
3. Bei Ausdrücken, die Klammern enthalten, werden die Klammern von innen nach außen aufgelöst.

5.5.8 Bezugnahme auf Merkmale

Der Assembler weist Namen und variablen Parametern Merkmale zu. Auf diese Merkmale kann man sich beziehen, um z.B. das Durchlaufen bestimmter Instruktionen vom Wert des entsprechenden Merkmals abhängig zu machen.

```
Format der Bezugnahme auf ein Merkmal        attr'par
```

attr Merkmalsbezeichnung (s.u.)
par Name oder
variabler Parameter

Jedes Merkmal hat eine bestimmte Bezeichnung, mit der es angesprochen wird.

Typenmerkmal	T
Längenmerkmal	L
Skalenfaktormerkmal	S
Ganzzahligkeitsmerkmal	I
Zählermerkmal	K
Anzahlmerkmal	N
Definitionsmerkmal	D

Die folgende Tabelle zeigt, auf welche Merkmale man sich für Namen und die verschiedenen variablen Parameter beziehen kann.

	Namen	symbolische Parameter	SET-Parameter	variable Systemparameter
T	X	X	X	X
L	X	X	X	X
S	X	X*	nur &SETC*	nur &SYSLIST(n)*
I	X	X	nur &SETC*	nur &SYSLIST(n)*
K		X	X	X
N		X	X	nur &SYSLIST(n) und &SYSLIST
D	X	X*	nur &SETC*	nur &SYSLIST(n)*

* nur zulässig, wenn der Wert ein Name ist

Tabelle 5-1 Bezugnahmen auf Merkmale von Namen und variablen Parametern

Merkmale von Namen

Der Wert des Merkmals wird an Hand der Daten berechnet, die der Name repräsentiert.

Dazu muß der Name definiert sein, d.h., er muß im Namenseintrag einer Assembleranweisung, bzw. eines Assemblerbefehls vorliegen oder im Operandeneintrag einer EXTRN- bzw. WXTRN-Anweisung. Die Instruktion, in der der Name definiert ist, muß im Assembler-Quellprogramm stehen. Der Name kann auch erst später im Quellprogramm definiert sein (siehe ASSEMBH, Benutzerhandbuch [1]; 'Lookahead-Mechanismus'). Eine Ausnahme ist hier die Bezugnahme auf das Definitionsmerkmal. Mit ihr wird abgefragt, ob ein Name zum Abfragezeitpunkt definiert ist.

Merkmale von variablen Parametern

Der Wert der Merkmale von variablen Parametern wird an Hand des aktuellen Wertes berechnet, der dem variablen Parameter zugewiesen wurde.

Die Merkmale von **SET-Parametern und variablen Systemparametern, außer &SYSLIST**, werden jeweils aus dem aktuellen Wert berechnet.

Die Merkmale von **symbolischen Parametern und dem variablen Systemparameter &SYSLIST** werden an Hand der Operanden des entsprechenden Makroaufrufs berechnet.

Werden symbolische Parameter oder &SYSLIST(n) über eine SETC-Anweisung im Makro neue Werte zugewiesen, so werden die Merkmale ebenfalls gemäß dieser aktuellen Werte berechnet.

Ist der Operand des Makroaufrufs ein Makroausdruck, dann erhält der entsprechende symbolische Parameter die Merkmale einer Zeichenkonstanten.

Ist der Operand eines inneren Makroaufrufs eine Operandenunterliste, können entweder die Merkmale der Unterliste oder jedes einzelnen Operanden der Unterliste bezeichnet werden. Das Typen-, Längen-, Ganzzahligkeits- und Skalenfaktormerkmal einer Unterliste ist dasselbe wie das entsprechende Merkmal des ersten Elementes aus der Unterliste.

5.5.8.1 T' Bezugnahme auf das Typenmerkmal

Das Typenmerkmal eines Namens oder eines variablen Parameters ist ein einzelner Buchstabe.

Die Bezugnahme auf das Typenmerkmal kann nur als Element von Zeichenausdrücken verwendet werden. Es muß stets alleine stehen.

Beispiele

```
Name      | Operation | Operanden
----------|-----------|-----------------
&A        | SETC      | T'&PAR
&B        | SETB      | (T'&X NE T'&Y)
.C        | AIF       | (T'&PAR NE 'F').D
```

- Die folgenden Typenmerkmale gelten für Namen und symbolische Parameter, die DC-, DS-, DXD- und CXD-Anweisungen bezeichnen. Den symbolischen Parametern muß hierbei im Makroaufruf ein entsprechender Name als Wert zugewiesen werden.

  A Adreßkonstante vom Typ A, implizite Länge, auf Wortgrenze ausgerichtet, CXD-Anweisung
  B Binärkonstante
  C Zeichenkonstante
  D Gleitpunktkonstante doppelter Genauigkeit, implizite Länge, auf Doppelwortgrenze ausgerichtet
  E Gleitpunktkonstante einfacher Genauigkeit, implizite Länge, ausgerichtet
  F Festpunktkonstante von Wortlänge, implizite Länge, auf Wortgrenze ausgerichtet
  G Festpunktkonstante, explizite Länge
  H Festpunktkonstante von Halbwortlänge, implizite Länge, auf Halbwortgrenze ausgerichtet
  K Gleitpunktkonstante, explizite Länge
  L Gleitpunktkonstante erweiterter Genauigkeit, implizite Länge, ausgerichtet
  P Dezimalkonstante, gepackt
  Q Relativadresse in einem externen Pseudoabschnitt
  R Adreßkonstante vom Typ A, S, V oder Y, explizite Länge
  S Adreßkonstante vom Typ S, implizite Länge, ausgerichtet
  V Adreßkonstante vom Typ V, implizite Länge, ausgerichtet
  X Sedezimalkonstante
  Y Adreßkonstante vom Typ Y, implizite Länge, ausgerichtet
  Z Dezimalkonstante, ungepackt

Beispiel

```
Name          | Operation | Operanden
--------------+-----------+-------------------------
* Makrodefinition
              | MACRO     |
              | EXTYP1    | &PAR
              | AIF       | (T'&PAR EQ 'F').FCON
              | .         |
              | .         |
.FCON         | DC        | ...
              | MEND      |

* Programm mit Makroaufruf
CONST         | DC        | F'3'
              | .         |
              | .         |
              | EXTYP1    | CONST
              | .         |
              | .         |
```

Dem symbolischen Parameter &PAR wird der Wert CONST zugewiesen.

```
Typ von CONST:  C
Typ von &PAR:   C
```

- Die folgenden Typenmerkmale gelten für Namen und symbolische Parameter,
  - die andere Instruktionen als DC-, DS-, DXD- und CXD-Anweisungen bezeichnen oder
  - die im Operandeneintrag einer EXTRN- oder WXTRN-Anweisung definiert sind.

  Auch in diesem Fall muß den symbolischen Parametern im Makroaufruf ein Name als Wert zugewiesen werden.

  I Assemblerbefehl
  J Name eines Programmabschnitts
  M Makroaufruf
  T Externer Name

Beispiel

```
Name        | Operation | Operanden
------------------------------------
* Makrodefinition
            | MACRO     |
            | EXTYP2    | &PAR
            | .         |
            | .         |
&PAR        | MVC       | ...
            | MEND      |

* Programmabschnitt mit Makroaufruf
PROG2       | START     |
            | EXTRN     | EXNAME
            | .         |
            | .         |
            | EXTYP2    | MNAME

* Generierte Instruktionen
PROG2       | START     |
            | EXTRN     | EXNAME
            | .         |
            | .         |
MNAME       | MVC       | ...

Typ von MNAME:   I
Typ von &PAR:    I
Typ von PROG2:   J
Typ von EXNAME:  T
```

Hinweis

Für alle bisher genannten Typenmerkmale gilt:
Namen, die mehrfach definiert sind, werden als Fehler gemeldet und erhalten das Typenmerkmal der ersten Definition.

- Die folgenden Typenmerkmale können nur symbolische Parameter haben, die im Makroaufruf nicht durch den Namen einer Instruktion ersetzt werden.

  N dem symbolischen Parameter wird im Makroaufruf zugewiesen:
  - ein selbstdefinierender Wert,
  - ein SETA-Parameter oder
  - ein SETB-Parameter

  O dem symbolischen Parameter wird im Makroaufruf kein Wert zugewiesen, der entsprechende Operand fehlt.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------
* Makrodefinition 1
            | MACRO     |
            | EXTYPO    | &PAR1,&PAR2
&A          | SETA      | 2
            | AIF       | (&PAR1 EQ 5).SYMO
            | .         |
            | .         |
.SYMO       | ANOP      |
            | EXTYPI    | &A                  (01)
            | MEND      |

* Makrodefinition 2
            | MACRO     |
            | EXTYPI    | &PARIN              (02)
            | AIF       | (T'&PARIN EQ 'N').SYMI
            | .         |
            | .         |
.SYMI       | ANOP      |
            | MEND      |

* Makroaufruf
            | EXTYPO    | 5                   (03)
```

(01) Aufruf von EXTYPI, dem symbolischen Parameter &PARIN aus der Musteranweisung des inneren Makro (02) wird der SETA-Parameter &A als Wert zugewiesen.

(03) Dem symbolischen Parameter &PAR1 wird der dezimale selbstdefinierende Wert 5 zugewiesen.

```
Typ von &PAR1:   N
Typ von &PARIN:  N
Typ von &PAR2:   O, fehlender Operand
```

- Das Typenmerkmal U (undefiniert) erhalten Namen oder variable Parameter, die zum Abfragezeitpunkt noch nicht definiert sind, bzw. die keinem der oben genannten Typen zugeordnet werden können.

  Speziell haben das Typenmerkmal U:

  – Namen,
    – die eine LTROG-Anweisung kennzeichnen oder
    – die eine EQU-Anweisung ohne dritten Operanden kennzeichnen,
  – SETC-Parameter, deren Wert kein Name ist,
  – variable Systemparameter, außer &SYSLIST(n),
  – Literale als Operanden von Makroaufrufen.

5.5.8.2 L’ Bezugnahme auf das Längenmerkmal

Das Längenmerkmal eines Namens oder eines variablen Parameters ist ein numerischer Wert, der die Länge des bezeichneten Speicherbereichs in byte wiedergibt (Beispiele siehe auch bei I’ Bezugnahme auf das Ganzzahligkeitsmerkmal).

Die Bezugnahme auf das Längenmerkmal kann nur als Element von arithmetischen Makroausdrücken verwendet werden.

Das Längenmerkmal ist 1 bei
- Namen oder variablen Parametern, die eine EQU-Anweisung ohne Längenangabe bezeichnen (siehe 4.2, EQU-Anweisung) und
- Namen oder variablen Parametern, deren Typenmerkmal J, T oder N ist.
- Verwendung von L’* in DC- und DS-Anweisungen, sowie in Literalen.

Das Längenmerkmal ist 0 bei
Namen oder variablen Parametern, deren Typenmerkmal M, O oder U ist.

Das Längenmerkmal von * (L’*) ist gleich der Länge der Instruktion, in der die Bezugnahme vorkommt.

Beispiele

```
Name    | Operation | Operanden
--------|-----------|-----------------
&A      | SETB      | (L’&B EQ 4)
&C      | SETA      | L’&X+30
```

5.5.8.3 S' Bezugnahme auf das Skalenfaktormerkmal

Das Skalenfaktormerkmal eines Namens oder variablen Parameters ist ein numerischer Wert, der die Zahl der Stellen angibt, die der gebrochene Teil von Festpunkt-, Gleitpunkt- und Dezimalkonstanten hat (Beispiele siehe bei I' Bezugnahme auf das Ganzzahligkeitsmerkmal).

Das Skalenfaktormerkmal kann nur für Namen abgefragt werden, die die genannten Konstantentypen bezeichnen und für symbolische Parameter oder SETC-Parameter, deren Wert der Name einer entsprechenden Konstanten ist.

Die Bezugnahme auf das Skalenfaktormerkmal darf nur als Element von arithmetischen Makroausdrücken vorkommen.

Das Skalenfaktormerkmal einer **Festpunkt- oder Gleitpunktzahl** entspricht dem Wert des Skalenfaktors.

Das Skalenfaktormerkmal einer **Dezimalzahl** ist die Anzahl der Ziffern rechts vom Dezimalpunkt.

5.5.8.4 I' Bezugnahme auf das Ganzzahligkeitsmerkmal

Das Ganzzahligkeitsmerkmal eines Namens oder variablen Parameters ist ein numerischer Wert, der sich auf den ganzzahligen Teil von Festpunkt-, Gleitpunkt- und Dezimalzahlen im Objektcode bezieht. Es wird errechnet aus Längen- und Skalenfaktormerkmal.

Die Bezugnahme auf das Ganzzahligkeitsmerkmal kann nur für Namen und symbolische Parameter abgefragt werden, die die genannten Konstantentypen bezeichnen und für SETC-Parameter, deren Wert der Name einer entsprechenden Konstanten ist.

Die Bezugnahme auf das Ganzzahligkeitsmerkmal darf nur als Element von arithmetischen Makroausdrücken vorkommen.

Berechnung des Ganzzahligkeitsmerkmals

Typenmerkmal	Ganzzahligkeitsmerkmal
H F G	I = 8*L-S-1
D E L K	I = 2*(L-1)-S
P	I = 2*L-S-1
Z	I = L-S

für alle Formeln gilt L = Längenmerkmal
S = Skalenfaktormerkmal

Beispiele

```
Name      Operation  Operanden
CONF1     DC         HS6'-15.75'          L = 2, S = 6, I = 9
CONF2     DC         FS8'100.3E-2'        L = 4, S = 8, I = 23
*
CONFL1    DC         ES2'46.415'          L = 4, S = 2, I = 4
CONFL2    DC         DS5'-3.729'          L = 8, S = 5, I = 9
*
COND1     DC         P'+1.25'             L = 2, S = 2, I = 1
COND2     DC         P'79.68'             L = 3, S = 2, I = 3
COND3     DC         Z'-543'              L = 3, S = 0, I = 3
COND4     DC         Z'79.68'             L = 4, S = 2, I = 2
```

5.5.8.5 K' Bezugnahme auf das Zählermerkmal

Das Zählermerkmal eines variablen Parameters ist ein numerischer Wert.

Die Bezugnahme auf das Zählermerkmal kann nur als Element von arithmetischen Makroausdrücken verwendet werden.

Zählermerkmal von symbolischen Parametern

Das Zählermerkmal eines symbolischen Parameters entspricht der Anzahl der Zeichen des entsprechenden Operanden aus dem Makroaufruf. Das Zählermerkmal eines fehlenden Operanden ist 0.

Ist der Operand eine Unterliste, dann schließt das Zählermerkmal die Klammern und Kommas der Unterliste mit ein. Man kann sich auf das Zählermerkmal jedes Suboperanden, unabhängig von der Schachtelungstiefe, beziehen (Beispiel siehe N' Bezugnahme auf das Anzahlmerkmal).

Enthält der Operand eines Makroaufrufs variable Parameter, dann entspricht das Zählermerkmal der Anzahl der Zeichen, nachdem die variablen Parameter durch ihre aktuellen Werte ersetzt wurden.

Zählermerkmal von SET-Parametern und variablen Systemparametern

SETA-Parameter	Anzahl der Zeichen, die benötigt werden, um den aktuellen Wert als Dezimalzahl ohne führende Nullen anzugeben.

Beispiele

```
&A1 SETA 111           K von &A1 = 3
&A2 SETA X'FF'         K von &A2 = 3
&A3 SETB (K'&A2 EQ 3)  Wert von &A3 = 1
```

SETB-Parameter	1
SETC-Parameter	Anzahl der Zeichen
&SYSDATE	9
&SYSECT	Anzahl der Zeichen
&SYSLIST(n[,m])	Anzahl der Zeichen
&SYSLIST	unzulässig
&SYSMOD	2
&SYSNDX	4
&SYSPARM	Anzahl der Zeichen
&SYSTEM	4
&SYSTIME	6
&SYSTSEC	Anzahl der Zeichen
&SYSVERM	6
&SYSVERS	6

5.5.8.6 N' Bezugnahme auf das Anzahlmerkmal

Das Anzahlmerkmal ist ein numerischer Wert und kann abgefragt werden für einen symbolischen Parameter, für einen SET-Parameter, oder für den gesamten Operandeneintrag eines Makroaufrufs.

Die Bezugnahme auf das Anzahlmerkmal kann nur als Element von arithmetischen Makroausdrücken verwendet werden.

Anzahlmerkmal eines symbolischen Parameters

Das Anzahlmerkmal eines symbolischen Parameters entspricht der Zahl der Suboperanden der entsprechenden Operandenunterliste aus dem Makroaufruf. Es ist möglich, sich auf das Anzahlmerkmal jedes Suboperanden der Operandenunterliste zu beziehen.

Über den variablen Systemparameter &SYSLIST können auch Stellungsoperanden eines Makroaufrufs angesprochen werden, die keinen entsprechenden symbolischen Parameter in der Musteranweisung haben.

Entspricht dem angesprochenen symbolischen Parameter keine Unterliste, sondern nur ein einzelner Operand, dann ist das Anzahlmerkmal 1.
Fehlt der Operand im Makroaufruf, ist das Anzahlmerkmal 0.

Anzahlmerkmal von SET-Parametern

Das Anzahlmerkmal eines SET-Parameters entspricht seiner aktuellen Dimension (siehe 6.2, Indizierte SET-Parameter).

Anzahlmerkmal von &SYSLIST

Das Anzahlmerkmal des variablen Systemparameter &SYSLIST entspricht der Anzahl der Stellungsoperanden im Operandeneintrag eines Makroaufrufs.

Beispiele

```
Name          | Operation | Operanden
--------------------------------------------------
* Musteranweisung
              | MAC       | &P1,&P2,&P3,&P4

* Makroaufruf
              | MAC       | 15,'NAME',ADR,(X,(Y,Z))
```

Bezugnahme auf (K'.., N'..)		Wert	K	N
&SYSLIST			-	4
&SYSLIST(1)	≙ &P1	15	2	1
&SYSLIST(1,2)	≙ &P1(2)	''	0	0
&SYSLIST(2)	≙ &P2	'NAME'	6	1
&SYSLIST(3)	≙ &P3	ADR	3	1
&SYSLIST(4)	≙ &P4	(X,(Y,Z))	9	2
&SYSLIST(4,1)	≙ &P4(1)	X	1	1
&SYSLIST(4,2)	≙ &P4(2)	(Y,Z)	5	2
&SYSLIST(4,2,1)	≙ &P4(2,1)	Y	1	1

5.5.8.7 D' Bezugnahme auf das Definitionsmerkmal

Das Definitionsmerkmal zeigt an, ob ein Name, der auch durch Ersetzung eines symbolischen Parameters entstanden sein kann, schon definiert ist oder noch nicht.

Die Bezugnahme auf das Definitionsmerkmal kann als Element in Vergleichsausdrücken, in logischen Ausdrücken und in arithmetischen Makroausdrücken verwendet werden.

Das Definitionsmerkmal hat den Wert 1 oder 0.

1 Der angesprochene Name oder symbolische Parameter ist definiert.
0 Der angesprochene Name oder symbolische Parameter ist noch nicht definiert.

Bei der Verwendung der Bezugnahme auf das Definitionsmerkmal in einem arithmetischen Makroausdruck werden die arithmetischen Werte +1 oder +0 eingesetzt.

Mit Hilfe der Bezugnahme auf das Definitionsmerkmal ist es z.B. möglich,
- in einer Schleifenverarbeitung abzufragen, ob ein Name schon definiert wurde und entsprechend die Definition zu durchlaufen oder nicht (siehe Beispiele, Makrodefinition 1);
- abzufragen, ob ein Name schon definiert wurde und davon abhängig, ob schon Bezug auf die Merkmale genommen werden kann (siehe Beispiele, Makrodefinition 2).

Beispiele

```
Name          | Operation | Operanden
--------------+-----------+-----------------------------------------------
* Makrodefinition 1
              | MACRO     |
              | .         |
              | .         |
              | AIF       | (D'NAME).SYM1
NAME          | DC        | F'0'
              | .         |
              | .         |
.SYM1         |           |
              | SR        | R1,R1
              | AIF       | (T'NAME EQ 'F').FCONST,(T'NAME EQ 'H').HCONST
              | IC        | R1,NAME
              | AGO       | .END
HCONST        | ANOP      |
              | LH        | R1,NAME
              | AGO       | .END
FCONST        | ANOP      |
              | L         | R1,NAME
.END          | MEND      |

* Makrodefinition 2
              | MACRO     |
              | .         |
              | .         |
              | AIF       | (D'NAME).SYM1
&TYP          | SETC      | 'U'
              | AGO       | .SYM2
              | .         |
              | .         |
.SYM1         | ANOP      |
&TYP          | SETC      | T'NAME
.SYM2         | ANOP      |
              | .         |
              | .         |
              | MEND      |
```

6 Variable Parameter

6.1 Symbolische Parameter

Symbolische Parameter werden in der Musteranweisung definiert und können dann im Namens-, Operations- und Operandeneintrag der Modellanweisungen einer Makrodefinition verwendet werden. Beim Aufruf des Makros müssen den symbolischen Parametern dann die jeweils aktuellen Werte zugewiesen werden. Ein symbolischer Parameter kann, außer in der Musteranweisung, mit Hilfe eines oder mehrerer variabler Parameter generiert werden (siehe 5.5.4, generierte variable Parameter).

Symbolische Parameter sind lokale Symbole. D.h., sie sind nur innerhalb einer Makrogenerierung bekannt, da ihnen jeweils im Makroaufruf neue Werte zugewiesen werden.

Kennwort- und Stellungsoperanden

Die symbolischen Parameter werden in der Musteranweisung im Namenseintrag und im Operandeneintrag definiert. Die Art des Operandeneintrags in der Musteranweisung bestimmt, ob es sich um Kennwort- oder Stellungsoperanden handelt (siehe Beispiel und 7.1.1, Stellungs- und Kennwortoperanden).

- **Kennwortoperanden** sind in der Musteranweisung durch das Gleichheitszeichen (=) kenntlich gemacht. Ihnen kann in der Musteranweisung ein Anfangswert zugewiesen werden. Im Makroaufruf werden die aktuellen Werte über das Kennwort zugewiesen. Die Reihenfolge der Kennwortoperanden im Makroaufruf ist beliebig.

  Das Kennwort im Makroaufruf ist der Name des variablen Parameters aus der Musteranweisung ohne das &-Zeichen, oder ein variabler Parameter, aus dem das Kennwort generiert wird.

- **Stellungsoperanden** werden auf Grund ihrer Stellung im Operandeneintrag beim Makroaufruf die aktuellen Werte zugewiesen. D.h., symbolische Parameter und die zugewiesenen aktuellen Werte müssen in der Musteranweisung und im Makroaufruf in der gleichen Reihenfolge stehen.

Regeln

- Ein symbolischer Parameter darf maximal aus 64 Zeichen bestehen.
- Das erste Zeichen muß ein kaufmännisches Und (&) sein.
- Symbolische Parameter dürfen nicht mit der Zeichenfolge &SYS beginnen (siehe auch 6.3, variable Systemparameter).
- Leerzeichen innerhalb eines symbolischen Parameters sind nicht zulässig.
- Derselbe variable Parameter darf nicht in derselben Makrodefinition als symbolischer Parameter und als SET-Symbol definiert werden.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------
* Makrodefinition
            | MACRO     |
&NAME       | MSYM      | &PAR1,&PARA=,&PARB=          (01)
&NAME       | EQU       | *
            | MVC       | &PARB,&PARA
            | B         | &PAR1
            | .         |
            | .         |
&PAR1       | EQU       | *
            | MEND      |

* Makroaufruf
BEGIN       | MSYM      | STOP,PARA=CONST,PARB=FIELD   (02)

* Generierte Instruktionen
BEGIN       | EQU       | *
            | MVC       | FIELD,CONST                  (03)
            | B         | STOP                         (04)
            | .         |
            | .         |
STOP        | EQU       | *                            (05)
```

(01) Musteranweisung; &PAR1 ist ein Stellungsoperand, &PARA und &PARB sind Kennwortoperanden.

(02) Makroaufruf; &PAR1 wird der aktuelle Wert STOP zugewiesen, &PARA über das Kennwort PARA der Wert CONST und &PARB über das Kennwort PARB der Wert FIELD.

(03), (04), (05)
In diesen Instruktionen werden die symbolischen Parameter durch die zugewiesenen aktuellen Werte ersetzt.

6.2 SET-Parameter

SET-Parameter sind variable Hilfsfelder, denen zur Übersetzungszeit mit Hilfe von SET-Anweisungen (siehe 7.2) Werte zugewiesen werden können. Sie werden entweder explizit mit einer GBLx- oder LCLx-Anweisung (siehe 7.2) definiert oder, bei lokalen SET-Parametern, implizit durch eine SET-Anweisung (siehe 7.2). Sie können im Namens-, Operations- und Operandeneintrag von Modellanweisungen verwendet werden. Durch die explizite Definition erhalten SET-Parameter den Anfangswert Null, bzw. leere Zeichenfolge. Diesen kann man mit Hilfe der SET-Anweisungen verändern und den SET-Parametern neue Feldinhalte zuweisen. Die SET-Parameter in den Modellanweisungen werden dann jeweils durch ihren aktuellen Wert ersetzt.

SET-Parameter können mit Hilfe eines oder mehrerer weiterer variabler Parameter generiert werden (siehe 5.3.3, generierte variable Parameter). Die Verwendung von SET-Parametern ist nicht auf Makros beschränkt. Die lokale oder globale Definition und der Einsatz solcher Hilfsfelder ist im ganzen Quellprogramm möglich (siehe Kap. 8).

Werden SET-Parameter explizit definiert, muß die Definition jeweils vor der ersten Verwendung durchlaufen werden. Wegen der Übersichtlichkeit empfiehlt es sich, alle Definitionen direkt nach der jeweiligen Musteranweisung aufzuführen.

Sowohl globale, als auch lokale SET-Parameter können einen arithmetischen, einen binären oder einen Zeichenwert als Feldinhalt haben. Diese Unterscheidung wird bereits in der GBLx-, bzw. LCLx-Anweisung festgelegt.

Das x in beiden Anweisungen kann durch folgende Zeichen ersetzt werden (siehe 7.2, GBLx- und LCLx-Anweisung):

A SET-Parameter mit einem arithmetischen Wert (SETA-Parameter)
B SET-Parameter mit einem binären Wert (SETB-Parameter)
C SET-Paramter mit einer Zeichenfolge als Wert (SETC-Parameter)

Regeln

- Ein SET-Parameter darf maximal aus 64 Zeichen bestehen.
- Das erste Zeichen muß ein kaufmännisches Und (&) sein.
- SET-Parameter dürfen nicht mit der Zeichenfolge &SYS beginnen (siehe auch 6.3, variable Systemparameter).
- Leerzeichen innerhalb von SET-Parametern sind nicht zulässig.

Programmierhinweis

Verwendet man in einer SET-Anweisung den Namen einer Variablen, die im Quellprogramm mit name DS CL(A-B) definiert ist, so meldet der ASSEMBH einen SEMANTIC ERROR (E35), wenn diese Definition nach dem Makroaufruf steht.
Steht diese Definition vor dem Makroaufruf oder wird statt dem geklammerten Ausdruck für CL der errechnete Wert eingesetzt, so akzeptiert der ASSMEMBH die SET-Anweisung.

Vordefinierte SET-Parameter

Den vordefinierten SET-Parametern können ohne vorherige Vereinbarung Werte zugewiesen werden.

SETA-Parameter,	global:	&AGm	
	lokal:	&ALm	mit $0 \leq m \leq 99$
SETB-Parameter,	global:	&BGn	
	lokal:	&BLn	mit $0 \leq n \leq 999$
SETC-Parameter,	nur global:	&CGm	mit $0 \leq m \leq 99$

Hinweis

Vordefinierte SET-Parameter werden von verschiedenen Tools benützt und vom ASSEMBH nur noch aus Kompatibilitätsgründen unterstützt. Für die Erstellung neuer Programme ist daher von ihrer Verwendung abzuraten.

Globale und lokale SET-Parameter

Globale SET-Parameter werden mit der GBLx-Anweisung definiert. Sie können zur Weitergabe von Werten zwischen Makrodefinitionen, bzw. Makrodefinition und Quellprogramm verwendet werden. Im zweiten Fall muß der Parameter auch im Quellprogramm definiert werden.

Ein globaler SET-Parameter mit seinem Wert ist dann in jeder Makrodefinition, bzw. im Quellprogramm bekannt, wenn er dort auch definiert worden ist. Durch die erneute Definition wird jedoch sein Anfangswert nicht wieder auf Null gesetzt.

Lokale SET-Parameter können mit der LCLx-Anweisung definiert werden. Sie sind dann nur in der jeweiligen Makrodefinition, bzw. im Quellprogramm, bekannt.

Wird der gleiche SET-Parameter in einer weiteren Makrodefinition ebenfalls als lokal definiert, dann gilt er in jeder Makrodefinition als eigener SET-Parameter. Sein jeweiliger Wert ist in anderen Makrodefinitionen, auch in einer inneren, nicht bekannt.

Bei lokalen SET-Parametern gibt es die Möglichkeit der impliziten Vereinbarung. Hier ist dann keine LCLx-Anweisung notwendig (siehe unten).

Beispiele

```
Name          | Operation | Operanden
--------------+-----------+-------------------------
* Makrodefinition 1
              | MACRO     |
&NAME         | LOAD1     |
              | GBLA      | &A                  (01)
*             |           |
&NAME         | LR        | 15,&A
&A            | SETA      | &A+1                (02)
              | MEND      |

* Makrodefinition 2
              | MACRO     |
              | LOAD2     |
              | GBLA      | &A                  (01)
*             |           |
              | LR        | 15,&A
&A            | SETA      | &A+1                (02)
              | MEND      |

* Makroaufrufe
BEGIN         | LOAD1     |
              | LOAD2     |
              | LOAD1     |
              | LOAD2     |

* Generierte Instruktionen
              | .         |
              | .         |
BEGIN         | LR        | 15,0
              | LR        | 15,1
              | LR        | 15,2
              | LR        | 15,3
              | .         |
              | .         |
```

(01) Definition des globalen SETA-Parameters &A; &A muß in jeder Makrodefinition definiert werden, in der der SETA-Parameter verwendet werden soll.

(02) Da &A als global definiert wurde, setzt die Addition nicht jedesmal auf dem Anfangswert 0 auf, sondern auf dem Ergebnis der vorangegangenen Addition, unabhängig davon, in welcher Makrodefinition diese vorkam.

Im folgenden Beispiel ist &A in beiden Makrodefinitionen als lokal definiert. Die übrigen Instruktionen sind dieselben wie im ersten Beispiel. Hier weist die Definition des SETA-Parameters bei jedem Aufruf des Makro den Anfangswert 0 zu, daher beeinflußt die SET-Anweisung den Wert von &A im LR-Befehl nicht mehr.

```
Name        | Operation | Operanden
------------+-----------+-----------------
* Makrodefinition 1
            | MACRO     |
&NAME       | LOAD1     |
            | LCLA      | &A
*           |           |
&NAME       | LR        | 15,&A
&A          | SETA      | &A+1
            | MEND      |

* Makrodefinition 2
            | MACRO     |
            | LOAD2     |
            | LCLA      | &A
*           |           |
            | LR        | 15,&A
&A          | SETA      | &A+1
            | MEND      |

* Makroaufrufe
BEGIN       | LOAD1     |
            | LOAD2     |
            | LOAD1     |
            | LOAD2     |

* Generierte Instruktionen
            | .         |
            | .         |
BEGIN       | LR        | 15,0
            | LR        | 15,0
            | LR        | 15,0
            | LR        | 15,0
            | .         |
            | .         |
```

Implizit vereinbarte lokale SET-Parameter

Lokale SET-Parameter gelten bereits als vereinbart, wenn sie im Namenseintrag einer SET-Anweisung auftreten. Der Assembler interpretiert jeden nicht vereinbarten SET-Parameter im Namenseintrag einer SET-Anweisung als lokalen SET-Parameter.

Implizit vereinbarte lokale SET-Parameter erhalten als Anfangswert den Wert, der im Operandeneintrag der SET-Anweisung spezifiziert ist. In diesem Fall muß im Operationseintrag der SET-Anweisung angegeben werden, um welche Art von SET-Parameter es sich handelt (siehe 7.2, SETA-, SETB- und SETC-Anweisung).

Indizierte SET-Parameter

Globale und lokale SET-Parameter können als indizierte SET-Parameter definiert werden.

`Format von indizierten SET-Parametern` `&par(d)`

&par ist der Name des SET-Parameters, unter dem die d Felder hintereinander angesprochen werden.

d Dimension; $1 \leq d \leq 2^{31}-1$

- In der Definition des SET-Parameters (GBLx oder LCLx) gibt d die Anzahl der Felder an, die unter dem Namen &par hintereinander stehen sollen. Hier kann d nur ein dezimaler selbstdefinierender Wert sein.
- Mit den SET-Anweisungen können den einzelnen Feldern Werte zugewiesen werden. Im Namenseintrag der SET-Anweisungen und bei der Verwendung in weiteren Modellanweisungen bezeichnet d das jeweilige Feld, das angesprochen werden soll.
  In den SET-Anweisungen kann ein Wert für d angegeben werden, der größer ist als der Wert von d aus der Definition. In diesem Fall wird der Wert von d aus der Definition durch den höheren Wert ersetzt und dieser gilt dann als Dimension des SET-Parameters.
  Hier ist d ein arithmetischer Makroausdruck.

Die Dimension eines SET-Parameters kann mit Hilfe des Anzahlmerkmals abgefragt werden:

`N'&par` ergibt die jeweils aktuelle Dimension.

Indizierte SET-Parameter können ebenso wie andere variable Parameter mit variablen Parametern oder mit alphanumerischen Zeichen verkettet werden.

Beispiel

```
Name          | Operation | Operanden
--------------+-----------+-----------------
* Definition
              | LCLC      | &PARAM(20)

* Zuweisung
&PARAM(1)     | SETC      | 'FIRST'
&PARAM(2)     | SETC      | 'SECOND'
              | .         |
              | .         |
```

6.3 Variable Systemparameter

Variablen Systemparametern werden vom Assembler Werte zugewiesen. Der Programmierer kann sie im Namens- Operations- und Operandeneintrag von Instruktionen verwenden, kann ihnen jedoch, außer bei &SYSMOD und &SYSLIST(n), keine neuen Werte zuweisen. Die variablen Systemparameter können mit Hilfe eines oder mehrerer variabler Parameter generiert werden (siehe 5.3.3, generierte variable Parameter).

Variable Systemparameter können mit alphanumerischen Zeichen verkettet werden. Bei ihrer Verwendung müssen die Syntaxregeln der entsprechenden Einträge eingehalten werden.

Es gibt variable Systemparameter mit lokalem oder mit globalem Charakter:

Lokale variable Systemparameter werden bei jedem Makroaufruf neu zugewiesen, sie sind also auf die einzelne Makrodefinition begrenzt. Sie können nur innerhalb von Makrodefinitionen verwendet werden.

Lokale variable Systemparameter sind:
- &SYSECT,
- &SYSLIST,
- &SYSNDX,
- &SYSTSEC und
- &SYSVERM.

Globale variable Systemparameter werden bei Beginn der Übersetzung zugewiesen und ihr Wert bleibt während der Übersetzung konstant.

Globale variable Systemparameter sind:
- &SYSDATE,
- &SYSPARM,
- &SYSTEM,
- &SYSTIME und
- &SYSVERS.

Der variable Systemparameter &SYSMOD hat hier eine Sonderstellung. Sein Standardwert wird bei Beginn der Übersetzung zugewiesen, er kann aber während des Programmablaufs geändert werden.

&SYSDATE

Der Wert von &SYSDATE ist das Datum der Übersetzung. Dieser Wert wird zu Beginn der Übersetzung berechnet und bleibt dann konstant.

```
Wert von &SYSDATE        mmttjjddd
```

mm	Monat
tt	Tag
jj	Jahr
ddd	Tagesnummer innerhalb des Jahres

Das Typenmerkmal von &SYSDATE ist U, das Zählermerkmal ist 9.

Beispiel

Im Beispiel ist das Datum der Übersetzung der 20. Oktober 1989.

```
Name        | Operation | Operanden
------------------------------------
* Makrodefinition
            | MACRO     |
            | MDATE     | &ENDE
            | .         |
            | .         |
            | B         | &ENDE
            | DC        | C'&SYSDATE'
            | DS        | 0H
&ENDE       | EQU       | *
            | MEND      |

* Makroaufruf
            | MDATE     | ENDE

* Generierte Instruktionen
            | .         |
            | .         |
            | B         | ENDE
            | DC        | C'102089293'
            | DS        | 0H
ENDE        | EQU       | *
            | .         |
            | .         |
```

&SYSECT

Der Wert von &SYSECT ist der Name des jeweils aktuellen Programmabschnitts zum Zeitpunkt des Makroaufrufs. Eine Zusatzinformation zu &SYSECT bietet der variable Systemparameter &SYSTSEC, der jeweils den Typ des entsprechenden Programmabschnitts enthält (siehe Beschreibung von &SYSTSEC).

Tritt &SYSECT in einer Makrodefinition auf, dann ist sein Wert der Name der letzten START-, CSECT-, DSECT-, XDSEC- oder COM-Anweisung, die vor dem Makroaufruf verarbeitet wurde. Hierbei wird nicht berücksichtigt, ob die Anweisung korrekt war, falls der Fehler nicht dazu geführt hat, daß der Programmabschnitt nicht definiert wurde.

Für die jeweilige Makrostufe ist der Wert von &SYSECT während der Verarbeitung einer Makrodefinition konstant, unabhängig von CSECT-, DSECT-, XDSEC- oder COM-Anweisungen oder von inneren Makroaufrufen.

Wenn dagegen innerhalb einer Makrodefinition CSECT-, DSECT-, XDSEC- oder COM-Anweisungen auftreten, beeinflussen sie den Wert von &SYSECT für alle folgenden inneren Makroaufrufe in dieser Makrodefinition und für alle folgenden Makroaufrufe einer anderen Makrostufe.

Das Typenmerkmal von &SYSECT ist U, das Zählermerkmal entspricht jeweils der Zahl der Zeichen, die den Wert von &SYSECT darstellen.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------
* Makrodefinition, äußerer Makro
            | MACRO     |
            | MACA      | &NAME,&CSECT
&NAME       | DC        | C'&SYSECT'
&CSECT      | CSECT     |
            | DC        | C'&SYSECT'
            | MACI      |
            | MEND      |

* Makrodefinition, innerer Makro
            | MACRO     |
            | MACI      |
            | DC        | C'&SYSECT'
            | MEND      |

* Programmabschnitt mit Makroaufrufen
PROG        | START     |
            | .         |
            | .         |
            | MACA      | FIRST,ACSECT
**************************************
            | MACA      | SECOND,BCSECT


* Generierte Instruktionen
FIRST       | DC        | C'PROG'           (01)
ACSECT      | CSECT     |
            | DC        | C'PROG'           (01)
            | DC        | C'ACSECT'         (02)
**************************************
SECOND      | DC        | C'ACSECT'         (03)
BCSECT      | CSECT     |
            | DC        | C'ACSECT'         (03)
            | DC        | C'BCSECT'         (04)
```

(01) Die beiden Anweisungen stehen in der gleichen Makrostufe, der Makro wurde in PROG aufgerufen.

(02) Die DC-Anweisung stammt aus dem Aufruf des inneren Makro MACI. Der Aufruf von MACI wird erst nach der CSECT-Anweisung ACSECT verarbeitet, daher ist der Wert von &SYSECT dann ACSECT.

(03) Die beiden Anweisungen stehen wieder in der gleichen Makrostufe, die letzte vorhergehende CSECT-Anweisung war aber ACSECT. Daher ist der Wert von &SYSECT in beiden Anweisungen ACSECT.

(04) Die Anweisung kommt wieder aus dem inneren Makroaufruf MACI. Das ist eine andere Makrostufe und daher nimmt &SYSECT den Namen der vorhergehenden CSECT-Anweisung BCSECT an.

&SYSLIST

Mit &SYSLIST kann man sich anstelle von symbolischen Parametern auf Stellungsoperanden von Makroaufrufen beziehen. Mit Hilfe des Index von &SYSLIST kann jeder Stellungsoperand, insbesondere jede Unterliste, in einem Makroaufruf angesprochen werden, auch wenn kein entsprechender symbolischer Parameter in der Musteranweisung definiert ist.

Auf diese Weise können auch die Merkmale der Operanden eines Makroaufrufs angesprochen werden.

```
Format von &SYSLIST        &SYSLIST(n[,m[,...]])
```

n bezeichnet den n-ten Stellungsoperanden eines Makroaufrufs.

Bei n = 0 erhält man den Namenseintrag des Makroaufrufs als Ergebnis.

m bezeichnet den m-ten Operanden derjenigen Operandenunterliste, die der n-te Operand eines Makroaufrufs ist.

Das Indizieren kann weiter ausgedehnt werden, um sich auf jeden Suboperanden einer Operandenunterliste zu beziehen (siehe 7.1.2, Operandenunterlisten).

n bzw. m kann jeder positive arithmetische Ausdruck sein, der als Operand einer SETA-Anweisung möglich ist.

Wenn für einen Wert von n oder m kein Operand vorliegt, erhält man als Ergebnis eine leere Zeichenfolge.

Nur bei der Bezugnahme auf das Anzahlmerkmal des Operandeneintrags in einem Makroaufruf (N'&SYSLIST, siehe 5.5.8, Bezugnahme auf Merkmale) kann &SYSLIST ohne den Index n, bzw. m verwendet werden. In diesem Fall erhält man als Ergebnis die Anzahl der Stellungsoperanden im Makroaufruf.

Die Bezugnahme auf das Typenmerkmal (T'&SYSLIST(n)) liefert den Typ des jeweils angesprochenen Operanden.

Beispiel

```
Name          | Operation | Operanden
--------------|-----------|-----------------------------------------
* Makrodefinition
              | MACRO     |
              | MLIST     | &PAR
&PAR          | DC        | C'&SYSLIST(2)'
              | DC        | C'&SYSLIST(3,2)'
              | DC        | C'&SYSLIST(4)'
              | DC        | C'&SYSLIST(2,1)'
              | DC        | C'&SYSLIST(3)'
              | MEND      |

* Makroaufruf
              | MLIST     | AAA,BBB,(C,D,,F),,H

* Generierte Instruktionen
AAA           | DC        | C'BBB'
              | DC        | C'D'
              | DC        | C''                    leere Zeichenfolge
              | DC        | C'BBB'
              | DC        | C'(C,D,,F)'
```

&SYSMOD

Mit &SYSMOD kann die Auflösung eines modusabhängigen Systemmakros abhängig vom Adressierungsmodus (siehe 3.3.3) gesteuert werden.

Bei Beginn der Übersetzung wird &SYSMOD der Standardwert 24 zugewiesen. Über den Systemmakro GPARMOD (siehe Einführung in die XS-Programmierung, Beschreibung [7]) kann der Wert von &SYSMOD geändert werden.

Für SYSMOD sind nur die Werte 24 und 31 zugelassen.

Das Typenmerkmal von &SYSMOD ist U, das Zählermerkmal ist 2.

Bei der Auflösung von modusabhängigen Systemmakros wird der Wert von &SYSMOD als Voreinstellung des globalen Parametermodus verwendet (siehe PARMOD und GPARMOD, Einführung in die XS-Programmierung, Beschreibung).

&SYSNDX

Der Wert von &SYSNDX ist ein Zähler, der bei jedem verarbeiteten inneren oder äußeren Makroaufruf innerhalb einer Übersetzungseinheit um 1 erhöht wird. Der Zähler hat vier Stellen, beim ersten Makroaufruf wird er auf 0001 gesetzt, beim zweiten auf 0002, usw.

Da der Wert von &SYSNDX keinen zulässigen Namen darstellt, muß &SYSNDX mit einem gültigen Namen verkettet werden, wenn &SYSNDX zur Generierung von Namen verwendet werden soll.

Der Wert von &SYSNDX ist während der Verarbeitung einer Makrodefinition konstant, unabhängig von eventuellen inneren Makroaufrufen. Man kann &SYSNDX deshalb z.B. dazu verwenden, eindeutige Namen für Instruktionsfolgen zu erzeugen, die durch mehrfache Aufrufe der gleichen Makrodefinition generiert wurden.

Steht &SYNDX als Element in einem arithmetischen Ausdruck, dann wird sein Wert arithmetisch interpretiert.

Das Typenmerkmal von &SYSNDX ist U, das Zählermerkmal ist 4.

Beispiel

```
Name          | Operation | Operanden
--------------+-----------+-------------------------------
* Makrodefinition, innerer Makro
              | MACRO     |
              | MACI      | &PARAM
A&SYSNDX      | SR        | 2,5
              | CR        | 2,5
              | BE        | B&PARAM
              | B         | A&SYSNDX
              | MEND      |

* Makrodefinition, äußerer Makro
              | MACRO     |
&NAME         | MACA      | &PARAM
&NAME         | SR        | 2,4
B&SYSNDX      | AR        | 2,6
              | MACI      | &PARAM
B&PARAM       | S         | 2,=F'1000'
A&SYSNDX      | ST        | 2,WORT
              | MEND      |

* 1. Aufruf von MACA
ALPHA         | MACA      | XXX                 (01)

* Generierte Instruktionen
ALPHA         | SR        | 2,4
B0106         | AR        | 2,6                 (02)
              | MACI      | XXX                 (04)
A0107         | SR        | 2,5                 (05)
              | CR        | 2,5
              | BE        | BXXX
              | B         | A0107               (06)
BXXX          | S         | 2,=F'1000'
A0106         | ST        | 2,WORT              (03)

* 2. Aufruf von MACA
BETA          | MACA      | YYY                 (07)

* Generierte Instruktionen
BETA          | SR        | 2,4
B0108         | AR        | 2,6                 (08)
              | MACI      | YYY                 (10)
A0109         | SR        | 2,5                 (11)
              | CR        | 2,5
              | BE        | BYYY
              | B         | A0109               (12)
BYYY          | S         | 2,=F'1000'
A0108         | ST        | 2,WORT              (09)
```

(01) Diese Instruktion soll der 106. Makroaufruf in dieser Übersetzungseinheit sein. A&SYSNDX wird ersetzt durch A0106 und B&SYSNDX durch B0106 in den Instruktionen (02) und (03).

(04) 107. Makroaufruf, A&SYSNDX wird ersetzt durch A0107, siehe Instruktionen (05) und (06).

(07) 108. Makroaufruf, A&SYSNDX und B&SYSNDX werden ersetzt durch A0108 und B0108, siehe Instruktionen (08) und (09).

(10) 109. Makroaufruf, A&SYSNDX wird ersetzt durch A0109, siehe Instruktionen (11) und (12).

&SYSPARM

Der Wert von &SYSPARM ist eine Folge von maximal 255 Zeichen. Diese wird in einer SDF-Option definiert (siehe ASSEMBH, Benutzerhandbuch [1]) und in der Makroauflösung ausgewertet. D.h., daß z.B. über eine Option, die von außen eingegeben wird, die Verarbeitung bestimmter Instruktionsfolgen gesteuert werden kann.

Das Typenmerkmal von &SYSPARM ist U, das Zählermerkmal entspricht der Anzahl der definierten Zeichen.

&SYSTEM

Der Wert von &SYSTEM ist die Betriebssystemversion unter der die Übersetzung läuft.

```
Wert von &SYSTEM          2vvv
```

2	steht für BS2000
vvv	Versionsbezeichnung des BS2000

Das Typenmerkmal von &SYSTEM ist U, das Zählermerkmal ist 4.

&SYSTIME

Der Wert von &SYSTIME ist die Tageszeit der Übersetzung. Dieser Wert wird zu Beginn der Übersetzung berechnet und bleibt dann konstant.

```
Wert von &SYSTIME         hhmmss
```

hh	Stunde
mm	Minute
ss	Sekunde

Das Typenmerkmal von &SYSTIME ist U, das Zählermerkmal ist 6.

&SYSTSEC

Der Wert von &SYSTSEC ist der Typ des jeweils aktuellen Programmabschnitts zum Zeitpunkt des Makroaufrufs. D.h., &SYSTSEC enthält jeweils den Typ des Programmabschnitts, dessen Name in &SYSECT abgelegt ist (siehe Beschreibung von &SYSECT).

Programmabschnitt	Wert von &SYSTSEC
START	CSECT
CSECT	CSECT
DSECT	DSECT
COM	COM
XDSEC	XDSEC

Tritt &SYSTSEC in einer Makrodefintion auf, dann ist sein Wert der Typ der letzten START-, CSECT-, DSECT-, COM- oder XDSEC-Anweisung, die vor dem Makroaufruf verarbeitet wurde. Hierbei wird nicht berücksichtigt, ob die Anweisung korrekt war, falls der Fehler nicht dazu geführt hat, daß der Programmabschnitt nicht definiert wurde.

Das Typenmerkmal von &SYSTSEC ist U, das Zählermerkmal entspricht jeweils der Zahl der Zeichen, die den Wert von &SYSTSEC bilden.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------------------
*Makrodefinition
            | MACRO     |
            | MACDSECT  | &NAME
&NAME       | DSECT     |
            | DS        | 4F
&SYSECT     | &SYSTSEC  |                  Zurückstellen auf den ursprüng-
            | MEND      |                  lichen Programmabschnitt

*Programmabschnitt mit Makroaufruf
PROG        | START     |
            | .         |
            | .         |
            | MACDSECT  | ADSECT
            | .         |
            | .         |

*Generierte Instruktionen
ADSECT      | DSECT     |
            | DS        | 4F
PROG        | CSECT     |                  Fortsetzung des 1. Programmab-
            | .         |                  schnitts
            | .         |
```

&SYSVERM

Der Wert von &SYSVERM ist die Versionsbezeichnung des Bibliotheksmakros in dem dieser Parameter verwendet wird.

Makrodefinitionen, die im Quellprogramm stehen, haben keine Versionsbezeichnung, daher ist hier die Verwendung von &SYSVERM nicht sinnvoll. Die Versionsbezeichnung wird in diesem Fall durch Leerzeichen ersetzt (VER␣␣␣).

```
Wert von &SYSVERM          VERvvv
```

vvv Versionsbezeichnung

Eine zu lange Versionsbezeichnung wird rechts abgeschnitten, eine zu kurze rechts mit Unterstrichen (_) aufgefüllt.

Das Typenmerkmal von &SYSVERM ist U, das Zählermerkmal ist 6.

Beispiele

Versionsbezeichnung	002	ergibt	VER002
Versionsbezeichnung	2	ergibt	VER2__
Versionsbezeichnung	1234	ergibt	VER123

&SYSVERS

Der Wert von &SYSVERS ist die Versionsbezeichnung des Quellprogramms, sofern dieses in einer Bibliothek steht.

Quellprogramme, die nicht aus einer Bibliothek eingelesen werden, haben keine Versionsbezeichnung, daher ist hier die Verwendung von &SYSVERS nicht sinnvoll. Die Versionsbezeichnung wird in diesem Fall durch Leerzeichen ersetzt (VER␣␣␣).

```
Wert von &SYSVERS          VERvvv
```

vvv Versionsbezeicnung

Eine zu lange Versionsbezeichnung wird rechts abgeschnitten, eine zu kurze rechts mit Unterstrichen (_) aufgefüllt (Beispiele siehe &SYSVERM).

Das Typenmerkmal von SYSVERS ist U, das Zählermerkmal ist 6.

7 Instruktionen der Makrosprache

7.1 Musteranweisung und Makroaufruf

Funktion

Die Musteranweisung gibt den Namen des Makro an und die symbolischen Parameter, die in dieser Makrodefinition vorkommen.

Der Makroaufruf ist eine Instruktion im Assembler-Quellprogramm. Durch die Verarbeitung des Makroaufrufs wird die Makrogenerierung angestoßen. Hierdurch werden die Instruktionen, die durch die Makrodefinition vorgegeben sind, unter Verwendung variabler Parameter in das Assembler-Programm eingefügt. Der Makroaufruf weist den symbolischen Parametern aus der Musteranweisung die jeweiligen aktuellen Werte zu.

Format der Musteranweisung

Name	Operation	Operanden
[&par]	name	[{pos_oper[,...] / keyw_oper[,...] / pos_oper[,...],keyw_oper[,...]}]

&par symbolischer Parameter
name Name des Makro

pos_oper Stellungsoperand, Format: &par
&par symbolischer Parameter

keyw_oper Kennwortoperand, Format: &par=[vala]
&par symbolischer Parameter
vala Anfangswert

In einer Musteranweisung können mehrere Stellungs- und Kennwortoperanden gemischt auftreten. Ihre Reihenfolge ist beliebig.

Format des Makroaufrufs

Name	Operation	Operanden
[{valn / .sym / &par1}]	{name / &par2}	[{pos_oper[,...] / keyw_oper[,...] / pos_oper[,...],keyw_oper[,...]}]

valn Name, der dem symbolischen Parameter im Namenseintrag der Musteranweisung zugewiesen werden soll

.sym Folgesymbol

&par1 variabler Parameter oder Verkettung von variablen Parametern und alphanumerischen Zeichen; hieraus muß der Namenseintrag generiert werden. Der Wert von &par1 wird dem symbolischen Parameter &par im Namenseintrag der Musteranweisung zugewiesen.

name Name des Makro

&par2 variabler Parameter oder Verkettung von variablen Parametern und alphanumerischen Zeichen; der Wert entspricht dem Makronamen.

pos_oper Stellungsoperand, Format: val

- val: Wert, der dem symbolischen Parameter aus der Musteranweisung zugewiesen werden soll oder variabler Parameter

keyw_oper Kennwortoperand, Format: par=val

- par: Kennwort, Name des symbolischen Parameters aus der Musteranweisung (ohne &-Zeichen)
- var: Wert, der dem symbolischen Parameter zugewiesen werden soll oder variabler Parameter

Im Makroaufruf können mehrere Stellungs- und Kennwortoperanden gemischt auftreten. Die Reihenfolge der Kennwortoperanden ist beliebig, die der Stellungsoperanden muß der Reihenfolge der Stellungsoperanden in der Musteranweisung entsprechen (Beispiel siehe 7.1.1).

Namenseintrag

Dem symbolischen Parameter im Namenseintrag der Musteranweisung wird der Namenseintrag des Makroaufrufs zugewiesen.

Enthält der Namenseintrag des Makroaufrufs einen variablen Parameter, muß dieser
- im Quellprogramm definiert sein, wenn der Makroaufruf im Quellprogramm steht, bzw.
- in der Makrodefinition definiert sein, in der der Makroaufruf steht.

Operationseintrag

Der Operationseintrag der Musteranweisung gibt den Namen an, mit dem der Makro aufgerufen werden muß. Dieser muß nach den Regeln für Namen gebildet sein (siehe 2.3, Namenseintrag). Bei einer Makrodefinition, die in einer Bibliothek vorliegt, muß er sich zusätzlich nach der LMS-Syntax richten (siehe LMS, Benutzerhandbuch [4]).

Liegen in einem Quellprogramm zwei oder mehr Makrodefinitionen mit demselben Namen im Operationseintrag vor, dann gilt die zuletzt eingelesene Makrodefinition als gültig bis zum Auftreten der nächsten Makrodefinition mit dem gleichen Namen.

Bei einer Makrodefinition im Quellprogramm ist es möglich, im Operationseintrag der Musteranweisung den mnemotechnischen Operationscode einer Assemblerinstruktion zu verwenden, ohne die Instruktion mit der OPSYN-Anweisung außer Kraft zu setzen. Die Definition des Makro muß vor seinem ersten Aufruf durchlaufen werden. In diesem Fall ist die ursprüngliche Funktion der Instruktion ausgeschaltet und jedes Auftreten dieser Instruktion gilt als Aufruf des entsprechenden Makros.
Man beachte, daß nach der Bearbeitung einer Makrodefinition im Quellprogramm mit dem Operationscode 'MACRO' in der Musteranweisung keine weiteren Makros mehr eingelesen werden.

Soll eine Makrodefinition in einer Bibliothek den mnemotechnischen Operationscode einer Assemblerinstruktion als Namen erhalten, muß der entsprechende Operationscode vorher mit der OPSYN-Anweisung außer Kraft gesetzt werden.

Enthält der Operationseintrag des Makroaufrufs einen variablen Parameter, muß dieser
- im Quellprogramm definiert sein, wenn der Makroaufruf im Quellprogramm steht, bzw.
- in der Makrodefintion definiert sein, in welcher der Makroaufruf vorkommt.

Operandeneintrag

Der Operandeneintrag der Musteranweisung bestimmt das Format und den Aufbau des Operandeneintrags vom zugehörigen Makroaufruf. Der Makroaufruf weist den symbolischen Parametern aus der Musteranweisung aktuelle Werte zu.

Wird im Makroaufruf ein Operand weggelassen, dann wird dem entsprechenden symbolischen Parameter als Zeichenwert der Nullstring (kein Zeichen) zugeordnet, bzw. der arithmetische Wert Null. Eine Ausnahme sind hier Kennwortoperanden, denen in der Musteranweisung ein Anfangswert zugewiesen wurde.

Regeln für Operanden von Makroaufrufen

- Ein Operand eines Makroaufrufs darf maximal 1020 Zeichen lang sein, auch nachdem eventuelle variable Parameter ersetzt wurden. Auch bei einem Kennwortoperanden darf der Wert 1020 Zeichen lang sein. Das Kennwort und das Gleichheitszeichen werden hier nicht mitgezählt.
- Hochkommas in einem Operanden müssen paarweise auftreten.

  Ein einzelnes Hochkomma innerhalb einer Zeichenfolge muß durch zwei Hochkommas dargestellt werden.

  Eine Bezugnahme auf das Längenmerkmal (L') eines Namens ist als Operand erlaubt.
- Linke und rechte Klammern müssen paarweise auftreten, außer sie stehen zwischen Hochkommas.

  Beginnt ein Operand mit einer linken Klammer, dann wird er als Unterliste interpretiert. Folgt auf die schließende rechte Klammer kein Leerzeichen oder Komma, wird der gesamte Operand als Zeichenfolge und nicht als Unterliste interpretiert.
- Stehen mehrere &-Zeichen hintereinander, muß ihre Anzahl geradzahlig sein, außer wenn ein &-Zeichen einen variablen Parameter kennzeichnet.
- Ein Komma bezeichnet das Ende eines Operanden oder eines Unterlisten-Elementes, außer es steht zwischen Hochkommas.
- Ist der Operand des Makroaufrufs ein Makroausdruck, dann erhält der entsprechende symbolische Parameter die Merkmale einer Zeichenkonstanten.
- Enthält ein Operand eines Makroaufrufs einen variablen Parameter als Wert, dann muß dieser
  - im Quellprogramm definiert sein, wenn der Makroaufruf im Quellprogramm steht, bzw.
  - in der Makrodefinition definiert sein, in der der Makroaufruf vorkommt.

7.1.1 Kennwort- und Stellungsoperanden

Kennwortoperanden sind in der Musteranweisung durch das Gleichheitszeichen (=) kenntlich gemacht. Ihnen kann in der Musteranweisung ein Anfangswert zugewiesen werden.

Im Makroaufruf werden Kennwortoperanden Werte über das Kennwort zugewiesen. Die Reihenfolge der Kennwortoperanden im Makroaufruf ist beliebig.

Es darf kein Komma angegeben werden, wenn ein Kennwortoperand im Makroaufruf weggelassen wird.

Bei einem Kennwortoperanden im Makroaufruf kann der Wert, der zugewiesen werden soll, durch einen weiteren variablen Parameter generiert werden.

Stellungsoperanden werden auf Grund ihrer Stellung im Operandeneintrag zugewiesen. Die symbolischen Parameter und die zugewiesenen aktuellen Werte müssen in der Musteranweisung und im Makroaufruf in der gleichen Reihenfolge stehen.

Soll einem Stellungsoperanden im Makroaufruf kein Wert zugewiesen werden, so wird er weggelassen und das folgende Trennkomma geschrieben. Werden die letzten Stellungsoperanden im Makroaufruf weggelassen, können auch die entsprechenden Kommas wegfallen.

Der Wert, der einem Stellungsoperanden im Makroaufruf zugewiesen werden soll, kann durch einen variablen Parameter generiert werden.

Im Makroaufruf können mehr Stellungsoperanden stehen als in der Musteranweisung angegeben wurden. In diesem Fall kann man sich nur über den variablen Systemparameter &SYSLIST auf die zusätzlichen Operanden beziehen (siehe 6.3).

Kennwort- und Stellungsoperanden können in der Musteranweisung und im Makroaufruf gemischt auftreten. Innerhalb der Stellungsoperanden muß aber die Reihenfolge in der Musteranweisung und im Makroaufruf übereinstimmen.

Beispiele

```
Name        | Operation | Operanden
------------------------------------------------
* Musteranweisung
&NAME       | MAC       | &P1,&P2=VAL2,&P3,&P4=,&P5

* Makroaufruf 1
VAL         | MAC       | P4=VAL4,VAL1,VAL3,VAL5
```

&P1,&P3 und &P5	Stellungsoperanden; ihnen werden die Werte VAL1, VAL3 und VAL5 zugewiesen.
&P2	Kennwortoperand mit dem Anfangswert VAL2
&P4	Kennwortoperand; der Wert VAL4 wird über den Makroaufruf zugewiesen.

```
* Makroaufruf 2
VAL         | MAC       | P4=VAL4,VAL1,,VAL5
```

In diesem Fall soll dem Stellungsoperanden &P3 kein Wert zugewiesen werden. Er ist im Makroaufruf durch das Komma ersetzt und erhält den Wert "leere Zeichenfolge".

7.1.2 Operandenunterlisten

Der Wert eines Stellungsoperanden beim Makroaufruf, bzw. der Wert eines Kennwortoperanden beim Makroaufruf oder in der Musteranweisung kann eine Operandenunterliste sein. Dabei gilt die gesamte Unterliste als ein Operand des Makroaufrufs und wird einem symbolischen Parameter aus der Musteranweisung zugeordnet.

```
Format der Operandenunterliste       (val,val[,...])
```

val — Wert oder neue Operandenunterliste

```
Bezugnahme auf Suboperanden der Operandenunterliste       &P(n[,m[,...]])
```

&P — symbolischer Parameter aus der Musteranweisung, dem die Operandenunterliste zugeordnet wurde

n — bezeichnet den n-ten Suboperanden der Operandenunterliste und ist ein positiver arithmetischer Makroausdruck

m — bezeichnet den m-ten Suboperanden der Unterliste, die ihrerseits den n-ten Suboperanden einer Unterliste darstellt und ist ein positiver arithmetischer Makroausdruck

Das Indizieren kann weiter ausgedehnt werden, um sich auf jeden Suboperanden innerhalb eines beliebigen Niveaus der Unterlisten zu beziehen.

Regeln

- Unterlisten müssen mit "(" beginnen und mit ")" enden; andernfalls wird der Operand (bzw. das Unterlistenelement) als Zeichenkette interpretiert.
- Operandenunterlisten können beliebig tief ineinander geschachtelt werden, wobei die Begrenzung nur durch die maximale Länge von Operanden gegeben ist.
- Die maximale Länge von 1020 Zeichen für Operanden von Makroaufrufen bezieht sich hier auf die gesamte Unterliste, einschließlich Suboperanden, Kommas und Klammern.
- Fehlende Suboperanden können ebenso wie bei den Stellungsoperanden durch ein Komma ersetzt werden und erhalten den Wert Null.
- Wird in einer Instruktion ein Operand eines Makroaufrufs, der keine Unterliste ist, als Unterliste angesprochen, dann bezieht sich &P(1) auf den ganzen Operanden. &P(n) mit n > 1 hat dann den Wert Null, bzw. leere Zeichenfolge.
- Die Suboperanden von Operandenunterlisten kann man bei Stellungsoperanden auch mit dem variablen Systemparameter &SYSLIST ansprechen (siehe 6.3).
- Enthält eine Operandenunterliste einen variablen Parameter im Suboperanden, muß dieser
  - im Quellprogramm definiert sein, wenn der Makroaufruf im Quellprogramm steht, bzw.
  - in der Makrodefinition definiert sein, in der der Makroaufruf vorkommt.
- Leerzeichen sind in Operandenunterlisten zulässig und werden als Bestandteil eines Suboperanden erkannt.
  z.B. hat die Unterliste (1␣,2,␣3) drei Suboperanden:

  Sub1 = 1␣
  Sub2 = 2
  Sub3 = ␣3

Beispiel

```
Name          | Operation | Operanden
--------------+-----------+-----------------------------------------
* Makrodefinition
              | MACRO     |
              | SUBEX     | &P
              | AIF       | (&P(3,2,1) EQ 'F' OR &P(5) EQ '').SYM
              | .         |
              | .         |
              | MEND      |

* Makroaufruf
              | SUBEX     | (A,(B,C),(D,(E,F)),G,)
```

Operand im Aufruf `(A,(B,C),(D,(E,F)),G,)`

Suboperanden			
	&P(1) = A		
	&P(2) = (B,C)	&P(2,1) = B &P(2,2) = C	
	&P(3) = (D,(E,F))	&P(3,1) = D &P(3,2) = (E,F)	&P(3,2,1) = E &P(3,2,2) = F
	&P(4) = G		
	&P(5) = ''		

In der AIF-Anweisung wird ersetzt &P(3,2,1) durch E
und &P(5) durch '' (leere Zeichenfolge)

7.1.3 Äußere und innere Makroaufrufe

Ein Makroaufruf, der innerhalb einer Makrodefinition als Modellanweisung verwendet wird, ist ein innerer Makroaufruf. Der Aufruf der entsprechenden Makrodefinition, die den inneren Aufruf enthält, wird als äußerer Makroaufruf bezeichnet. Der äußere Makroaufruf kann im Quellprogramm stehen oder seinerseits innerer Makroaufruf in einer weiteren Makrodefinition sein.

Ein Makroaufruf im Quellprogramm wird auch als 1. Stufe bezeichnet. Der innere Makroaufruf in der zugehörigen Makrodefinition als 2. Stufe. Die zum Aufruf der 2. Stufe gehörende Makrodefinition kann wieder einen inneren Makroaufruf enthalten, der dann der 3. Stufe entspricht, usw.

Auf diese Weise können maximal 255 Stufen geschachtelt werden. Die mögliche Schachtelungstiefe hängt zusätzlich von der Komplexität der Makrodefinitionen und vom verfügbaren Speicherplatz ab.

Weitergabe von Werten über innere Makroaufrufe

Die Werte im Operandeneintrag des inneren Makroaufrufs ersetzen die symbolischen Parameter der zugehörigen Musteranweisung.

Es ist möglich, Werte von einem äußeren Makro über einen inneren Makroaufruf an die entsprechende zugehörige Makrodefinition weiterzugeben.

Dazu müssen entweder
- die symbolischen Parameter des äußeren Makro oder
- im äußeren Makro definierte SET-Parameter

als Operanden des inneren Makroaufrufs verwendet werden.

Die Werte, die im äußeren Makroaufruf als Operanden angegeben werden, ersetzen dann jeweils die entsprechenden symbolischen Parameter.

Beispiel

```
Name          | Operation | Operanden
--------------+-----------+-------------------------------------
* Makrodefinition 1
              | MACRO     |
&NAME         | MADD      | &ART,&F1,&F2,&F3,&F4            (02)
              | .         |
              | .         |
&NAME         | MPACK     | &F1,&F2,&F3,&F4                 (03)
              | .         |
              | .         |
&NAME         | AP        | &F1,&F2
              | AP        | &F1,&F3
              | AP        | &F1,&F4
              | MEND      |

* Makrodefinition 2
              | MACRO     |
&NAME         | MPACK     | &A1,&A2,&A3,&A4                 (04)
&NAME         | PACK      | &A1,&A1
              | PACK      | &A2,&A2
              | PACK      | &A3,&A3
              | PACK      | &A4,&A4
              | MEND      |

* äußerer Makroaufruf
UNP           | MADD      | U,FIELD1,FIELD2,FIELD3,FIELD4   (01)

* Generierter innerer Makroaufruf
UNP           | MPACK     | FIELD1,FIELD2,FIELD3,FIELD4

* Generierte Instruktionen
              | .         |
              | .         |
UNP           | AP        | FIELD1,FIELD2
              | AP        | FIELD1,FIELD3
              | AP        | FIELD1,FIELD4
              | .         |
              | .         |
FIELD1        | DC        | CL6'000010'
              | .         |
              | .         |
```

(01) Äußerer Makroaufruf;

(02) Musteranweisung von Makrodefinition 1; die symbolischen Parameter werden von den Werten aus dem Makroaufruf ersetzt.

(03) Innerer Makroaufruf; an die symbolischen Parameter in diesem Makroaufruf werden die Werte des äußeren Makroaufrufs übergeben.

(04) Musteranweisung von Makrodefinition 2; diese symbolischen Parameter werden ersetzt von denen aus dem Makroaufruf (03) und damit von den übergebenen Werten.

7.1.4 Alternatives Anweisungsformat

Das alternative Anweisungsformat ist eine zusätzliche Möglichkeit der Formatdarstellung von Musteranweisung und Makroaufruf. Es kann außerdem für die LCLx- und GBLx-Anweisung, sowie für Format 2 der AIF- und AGO-Anweisung eingesetzt werden.

Format

Name	Operation	Operanden	Fortsetzungszeichen
[&par]	name	operand, operand[, ...]	x x

Namenseintrag siehe Beschreibung des normalen Formats
Operationseintrag siehe Beschreibung des normalen Formats
operand Stellungs- oder Kennwortoperand(en) bzw. Suboperanden bei Operandenunterlisten.
x irgendein Zeichen ungleich dem Leerzeichen

Beschreibung

Das alternative Anweisungsformat ermöglicht es, in jeder Zeile einen oder mehrere Operanden zu schreiben und in der nächsten Zeile mit dem nächsten Operanden fortzusetzen. Entsprechendes gilt für Suboperanden bei Operandenunterlisten.

Für den Operandeneintrag gilt:

- Damit der Operandeneintrag in der nächsten Zeile, beginnend in der Fortsetzungsspalte, fortgesetzt werden kann, muß dem vorangehenden Operanden ein Komma folgen. Die Fortsetzungszeichenspalte muß dann ein Zeichen ungleich dem Leerzeichen enthalten.
- Nach dem Komma und einem Leerzeichen können Bemerkungen bis einschließlich zur Endspalte geschrieben werden.
- Ein Leerzeichen nach einem Operanden (kein Suboperand) zeigt das Ende des Operandeneintrags an.
- Auch nach dem Ende des Operandeneintrags können Bemerkungen folgen. Steht hier in der Fortsetzungszeichenspalte ein Zeichen ungleich dem Leerzeichen, wird der Bemerkungseintrag in der nächsten Zeile, beginnend in der Fortsetzungsspalte, fortgesetzt.

Beispiel

```
Name        Operation   Operanden                    Fortsetzungszeichen
                        OP1   OPER1,OPER2,OPER3      COMMENT                (01)
*
                        OP2A  OPER1,                 COMMENT  X
                              OPER2,                 COMMENT  X
                              OPER3                  COMMENT
*                                                                          (02)
                        OP2B  OPER1,                 COMMENT  X
                              OPER2,OPER3,OPER4,     COMMENT  X
                              OPER5                  COMMENT
*
                        OP3   OPER1,                 COMMENT  X
                              OPER2                  COMMENT  X            (03)
                              COMMENT
*
                        OP4   OPER1,OPER2            COMMENT  X
                              (SUBOPER31,            COMMENT  X            (04)
                              SUBOPER32)
```

(01) Normales Format
(02) Alternatives Anweisungsformat mit Bemerkungseintrag
(03) Alternatives Anweisungsformat mit Bemerkungseintrag, der in der nächsten Zeile fortgesetzt wird.
(04) Alternatives Anweisungsformat mit Operandenunterlisten

7.2 Beschreibung der Makroanweisungen

ACTR Verzweigungen zählen

Funktion

Die ACTR-Anweisung dient zur Begrenzung der AGO- und AIF-Verzweigungen, die in einer Makrodefinition oder im Assembler-Quellprogramm verarbeitet werden. Hierdurch können z.B. Endlosschleifen bei der Makrogenerierung vermieden werden.

Format

Name	Operation	Operanden
[.sym]	ACTR	arexp

.sym Folgesymbol
arexp positiver arithmetischer Makroausdruck

Beschreibung

Die ACTR-Anweisung setzt einen Zähler auf den Wert, den arexp angibt. Bei jeder AGO- oder AIF-Verzweigung wird der Zähler um 1 vermindert.

Die ACTR-Anweisung kann für jede Makrodefinition und für das Assembler-Quellprogramm gesetzt werden. Die verschiedenen Zähler sind voneinander unabhängig.

Ist der Zähler vor der Verzweigung Null, geschieht folgendes:

- Bei einem Zähler, der für eine Makrodefinition gilt, wird die Verarbeitung dieser Makrodefinition und evtl. enthaltenen innerer Makrodefinitionen abgebrochen und die nächste Instruktion nach dem Makroaufruf verarbeitet.
- Bei einem Zähler, der für das Assembler-Quellprogramm gilt, wird die Übersetzung abgebrochen und nur die bisher gefundenen Fehler gemeldet.

Programmierhinweis

Ist keine ACTR-Anweisung angegeben, nimmt der Assembler als Wert des Zählers 4096 an.

Beispiel siehe AGO-Anweisung

AIF Bedingt verzweigen

Funktion

Die AIF-Anweisung ermöglicht eine Verzweigung abhängig von Bedingungen.

Format 1

Name	Operation	Operanden
[.sym1]	AIF	(logexp).sym2

.sym1 Folgesymbol im Standardformat
.sym2 Folgesymbol im Standardformat oder generiert
logexp logischer Ausdruck

Hinweis

Der Operationscode AIFB, der wie AIF verwendet werden kann, wird nur noch aus Kompatibilitätsgründen unterstützt.

Beschreibung

logexp wird nach den Regeln für logische Ausdrücke berechnet, das Ergebnis kann wahr oder falsch sein:

- Ist logexp wahr, dann wird zu der Instruktion verzweigt, die durch .sym2 bezeichnet ist.
- Ist logexp falsch, dann wird die der AIF-Anweisung folgende Instruktion bearbeitet.

Programmierhinweise

1. Mit der AIF-Anweisung kann vorwärts oder rückwärts verzweigt werden. Durch die Verzweigung übersprungene Instruktionen werden dabei nicht generiert.
2. Mit der AIF-Anweisung kann nicht von einer Makrodefinition in eine andere oder vom Quellprogramm in eine Makrodefinition, bzw. umgekehrt, verzweigt werden.

Beispiel

Das Beispiel zeigt einen Makro, in dem die symbolischen Parameter aus dem Makroaufruf nach bestimmten Bedingungen abgefragt werden. Nur, wenn alle geforderten Bedingungen erfüllt sind, wird die Bearbeitung durchgeführt.

```
Name        Operation   Operanden
            MACRO
&NAME       TAIF        &REG,&ADR,&NR                          (01)
            LCLB        &ERRIN                                 (02)
            .
            .
            AIF         (&REG GE 2 AND &REG LE 13).OK1
            MNOTE       0,'INCORRECT REG &REG'
&ERRIN      SETB        1
.OK1        AIF         (&ADR NE '').OK2
            MNOTE       0,'MISSING ADR'                        (03)
&ERRIN      SETB        1
.OK2        AIF         (&NR GE 1 AND &NR LE 4).OK3
            MNOTE       0,'INCORRECT NR &NR'
&ERRIN      SETB        1
.OK3        ANOP
            AIF         (&ERRIN).MEND                          (04)
            .
            .
.MEND       MEND
```

(01) Musteranweisung mit symbolischen Parametern &REG, &ADR und &NR

(02) Definition eines "Fehlerindikators" &ERRIN

(03) Die 3 AIF-Abfragen prüfen, ob die symbolischen Parameter entsprechend den geforderten Bedingungen im Makroaufruf angegeben wurden. Ist der Parameter richtig, wird zur nächsten Abfrage verzweigt.
Im Fehlerfall wird die MNOTE-Anweisung durchlaufen, die die entsprechende Erklärung ausgibt, und der Fehlerindikator gesetzt.

(04) Abfrage, ob an irgendeiner Stelle der Fehlerindikator gesetzt wurde. Falls ja, wird zum Ende verzweigt, falls nicht, bearbeitet der Assembler die folgenden Instruktionen.

Format 2

Name	Operation	Operanden
[.sym]	AIF	{(logexp).sym1}[,...]

.sym Folgesymbol im Standardformat
logexp logischer Ausdruck
.sym1 Folgesymbol im Standardformat oder generiert

Beschreibung

Format 2 der AIF-Anweisung entspricht n AIF-Anweisungen hintereinander. Die logischen Ausdrücke werden nacheinander berechnet und zum ersten Folgesymbol verzweigt, dessen zugehöriger logischer Ausdruck wahr ist.

Programmierhinweis

Für Format 2 der AIF-Anweisung ist das alternative Anweisungsformat möglich (siehe 7.1.4).

Beispiel

```
Name    Operation   Operanden                          Fortsetzungszeichen
        AIF         ('&S' EQ '+').PLUS,                X
                    ('&S' EQ '-').MINUS,               X
                    ('&S' EQ '=').EQ
        .
        .
```

Das Beispiel ist im alternativen Anweisungsformat geschrieben. Abhängig vom Inhalt von &S wird zum jeweiligen Folgesymbol verzweigt.

AGO Unbedingt verzweigen

Funktion

Die AGO-Anweisung ermöglicht eine Verzweigung unabhängig von Bedingungen.

Format 1

Name	Operation	Operanden
[.sym1]	AGO	.sym2

.sym1 Folgesymbol im Standardformat
.sym2 Folgesymbol im Standardformat oder generiert.

Hinweis

Der Operationscode AGOB, der wie AGO verwendet werden kann, wird nur noch aus Kompatibilitätsgründen unterstützt.

Beschreibung

.sym2 bezeichnet die nächste Instruktion, die vom Assembler verarbeitet werden soll.

Programmierhinweise

1. Mit der AGO-Anweisung kann vorwärts oder rückwärts verzweigt werden. Durch die Verzweigung übersprungene Instruktionen werden dann nicht generiert.
2. Mit der AGO-Anweisung kann nicht von einer Makrodefinition in eine andere oder vom Quellprogramm in eine Makrodefinition bzw. umgekehrt, verzweigt werden.

Beispiel

Das Beispiel zeigt die Verwendung von AGO in einer Schleife in Zusammenhang mit einer AIF-Abfrage.

```
Name          | Operation | Operanden
--------------------------------------------------
* Makrodefinition
              | MACRO     |
              | REG       | &PRE
              | LCLA      | &R
              | .         |
              | .         |
              | ACTR      | 100                  (01)
.LOOP         | ANOP      |                      (02)
&PRE&R        | EQU       | &R
&R            | SETA      | &R+1
              | AIF       | (&R GT 15).END       (03)
              | AGO       | .LOOP                (04)
              | .         |
              | .         |
.END          | MEND      |

* Makroaufruf
              | REG       | REG

* Generierte Instruktionen
REG0          | EQU       | 0
REG1          | EQU       | 1
REG2          | EQU       | 2
              | .         |
              | .         |
```

(01) "Notbremse", falls das Schleifenendekriterium nicht erreicht wird
(02) Schleifenanfang
(03) Abfrage auf das Endekriterium und Sprung aus der Schleife
(04) Schleifenende und Rücksprung zum Anfang

Format 2

Name	Operation	Operanden
[.sym]	AGO	(arexp){.sym1}[,...]

.sym Folgesymbol im Standardformat
arexp positiver arithmetischer Makroausdruck
.sym1 Folgesymbol im Standardformat oder generiert

Beschreibung

arexp gibt die Nummer des Folgesymbols im Operandeneintrag an, zu dem verzweigt werden soll.

Ist das Ergebnis von arexp größer als die Anzahl der Folgesymbole oder kleiner als 1, dann wird keine Verzweigung durchgeführt.

Programmierhinweis

In Format 2 der AGO-Anweisung ist das alternative Anweisungsformat möglich (siehe 7.1.4).

Beispiel

Name	Operation	Operanden
	AGO . .	(&EX).LOOP1,.LOOP2,.LOOP3

Hier wird zu .LOOP2 verzweigt, wenn &EX den Wert 2 annimmt.
Hat &EX einen Wert, der größer ist, als 3, dann wird die Instruktion ausgeführt, die auf die AGO-Anweisung folgt.

Im alternativen Anweisungsformat wird das gleiche Beispiel folgendermaßen codiert:

Name	Operation	Operanden	Fortsetzungszeichen
	AGO . .	(&EX).LOOP1, .LOOP2, .LOOP3	X X

ANOP Nulloperation

Funktion

Mit der ANOP-Anweisung können Folgesymbole als Sprungziel für eine Verzweigung definiert werden. Sie bewirkt keine Funktionsausführung.

Format

Name	Operation	Operanden
[.sym]	ANOP	

.sym Folgesymbol

Beschreibung

Die ANOP-Anweisung bewirkt selbst keine Operation, sondern dient nur als Ziel für Verzweigungen in der Makrosprache. Der Assembler generiert jeweils die nächste folgende Instruktion.

Programmierhinweise

1. Die ANOP-Anweisung wird vor allem eingesetzt, wenn zu einer Instruktion verzweigt werden soll, die kein Folgesymbol im Namensfeld erlaubt. In diesem Fall muß vor der entsprechenden Instruktion die ANOP-Anweisung eingefügt werden.
2. Durch Verwendung von ANOP bei Sprungzielen kann ohne Probleme ein weiterer Befehl als erster eingefügt oder gestrichen werden.

Beispiel

```
Name      Operation   Operanden
          AGO         .SYM
          .
          .
.SYM      ANOP                    einfügen
                                  <-------          AIF   (D'RDEF).SYM1
RDEF      DC          F'123'
                                  <------- .SYM1    ANOP
          AIF         (&A EQ 0).SYM2
          .
          .
```

GBLx Globalen SET-Parameter definieren

Funktion

Die GBLx-Anweisung definiert einen oder mehrere globale SET-Parameter.

Format

Name	Operation	Operanden
	GBL{A|B|C}	{&par | &par(d)}[,...]

&par	globaler SET-Parameter
&par(d)	indizierter globaler SET-Parameter
d	Dimension; dezimaler selbstdefinierender Wert
GBLA	Definition eines SETA-Parameters
GBLB	Definition eines SETB-Parameters
GBLC	Definition eines SETC-Parameters

Beschreibung

Die GBLx-Anweisung definiert den, bzw. die globalen SET-Parameter im Operandeneintrag und weist ihnen ggf. einen Anfangswert zu. Die definierten SET-Parameter haben alle den Typ, der im Operationseintrag angegeben ist.

Anfangswerte:	SETA-Parameter	0
	SETB-Parameter	0
	SETC-Parameter	leere Zeichenkette

Programmierhinweise

1. Einem globalen SET-Parameter wird nur von der ersten verarbeiteten GBLx-Anweisung ein Anfangswert zugewiesen. Nachfolgende GBLx-Anweisungen in einer anderen Makrodefiniton oder im Quellprogramm bedeuten nur die Definiton des SET-Paramters und weisen nicht wieder den Anfangswert zu.
2. SET-Parameter dürfen nicht mit der Zeichenfolge &SYS beginnen (siehe 6.3, Variable Systemparameter).
3. Für die GBLx-Anweisung ist das alternative Anweisungsformat möglich (siehe 7.1.4).

Beispiel

```
Name        | Operation | Operanden
------------|-----------|-----------------------------
            | GBLA      | &(&AR1),&AR2(20),&AR3,&AR4
```

&(&AR1) globaler SETA-Parameter, dessen Name mit Hilfe von &AR1 generiert werden soll
&AR2(20) indizierter globaler SETA-Parameter mit der Dimension 20
&AR3, &AR4 globale SETA-Parameter

Im alternativen Anweisungsformat kann das gleiche Beispiel folgendermaßen codiert werden:

```
Name        | Operation | Operanden          | Fortsetzungszeichen
------------|-----------|--------------------|--------------------
            | GBLA      | &(&AR1),           | X
            |           | &AR2(20),          | X
            |           | &AR3,              | X
            |           | &AR4               |
```

LCLx Lokalen SET-Parameter definieren

Funktion

Die LCLx-Anweisung definiert einen oder mehrere lokale SET-Parameter.

Format

Name	Operation	Operanden
	LCL[A|B|C]	{&par | &par(d)}[,...]

&par	lokaler SET-Parameter
&par(d)	indizierter lokaler SET-Parameter;
d	Dimension; dezimaler selbstdefinierender Wert
LCLA	Definition eines SETA-Parameters
LCLB	Definition eines SETB-Parameters
LCLC	Definition eines SETC-Parameters

Beschreibung

Die LCLx-Anweisung definiert den, bzw. die lokalen SET-Parameter im Operandeneintrag und weist ihnen einen Anfangswert zu.

Die definierten SET-Parameter haben alle den Typ, der im Operationseintrag angegeben ist.

Anfangswerte:	SETA-Parameter	0
	SETB-Parameter	0
	SETC-Parameter	leere Zeichenkette

Programmierhinweise

1. Wird derselbe SET-Parameter sowohl im Quellprogramm als auch in einer oder mehreren Makrodefinitionen als lokal definiert, so gilt er jeweils als neuer SET-Parameter mit dem Anfangswert.
2. SET-Parameter dürfen nicht mit der Zeichenfolge &SYS beginnen (siehe 6.3, Variable Systemparameter).
3. Für die LCLx-Anweisung ist das alternative Anweisungsformat möglich (siehe 7.1.4).

Beispiel

```
Name        | Operation | Operanden
------------|-----------|-----------------------------
            | LCLA      | &(&AR1),&AR2(20),&AR3,&AR4
```

&(&AR1) lokaler SETA-Parameter, dessen Name mit Hilfe von &AR1 generiert werden soll
&AR2(20) indizierter lokaler SETA-Parameter mit der Dimension 20
&AR3, &AR4 lokale SETA-Parameter

Im alternativen Anweisungsformat kann das gleiche Beispiel folgendermaßen codiert werden:

```
Name        | Operation | Operanden          | Fortsetzungszeichen
------------|-----------|--------------------|--------------------
            | LCLA      | &(&AR1),           | X
            |           | &AR2(20),          | X
            |           | &AR3,              | X
            |           | &AR4               |
```

MACRO Anfangsanweisung

Format

Name	Operation	Operanden
	MACRO	

Beschreibung

Die MACRO-Anweisung kennzeichnet den Anfang einer Makrodefinition. Sie muß die erste Instruktion in jeder Makrodefinition sein.

MEND Endanweisung

Format

Name	Operation	Operanden
[.sym]	MEND	

.sym Folgesymbol

Beschreibung

Die MEND-Anweisung kennzeichnet das Ende einer Makrodefinition. Sie muß die letzte Instruktion in jeder Makrodefinition sein.

Das Folgesymbol .sym dient als Sprungziel für AIF- oder AGO-Anweisungen.

MEXIT Ausgang aus einer Makrodefinition definieren

Funktion

Die MEXIT-Anweisung definiert einen Ausgang aus einer Makrodefinition. Bei der Generierung wird dadurch die aktuelle Makrogenerierung beendet.

Format

Name	Operation	Operanden
[.sym]	MEXIT	

.sym Folgesymbol

Beschreibung

Auf Grund der MEXIT-Anweisung beendet der Assembler die aktuelle Makrogenerierung und verarbeitet die Instruktion, die unmittelbar auf den entsprechenden Makroaufruf folgt. Steht der Makroaufruf im Quellprogramm, wird die nächstfolgende Quellprogramm-Instruktion verarbeitet. Ist der Makroaufruf Modellanweisung in einer äußeren Makrodefinition, dann wird die nächstfolgende Modellanweisung verarbeitet.

Das Folgesymbol .sym dient als Sprungziel für AIF- oder AGO-Anweisungen.

Programmierhinweise

1. Die MEXIT-Anweisung darf nur innerhalb von Makrodefinitionen verwendet werden.
2. Die MEXIT-Anweisung darf nicht mit der MEND-Anweisung verwechselt werden. Die MEND-Anweisung muß immer die letzte Anweisung einer Makrodefinition sein, auch wenn diese eine oder mehrere MEXIT-Anweisungen enthält.

Beispiel

```
Name      | Operation | Operanden
          | MACRO     |
          | .         |
          | .         |
          | LCLB      | &OKIN1,&OKIN2
          | .         |
          | .         |
          | AIF       | (&OKIN1).OK1        (01)
          | MNOTE     | 'NO OKIN1'
          | MEXIT     |                     (02)
.OK1      | AIF       | (&OKIN2).OK2        (01)
          | MNOTE     | 'NO OKIN2'
          | MEXIT     |                     (02)
          | .         |
          | .         |
          | MEND      |
```

(01) Abfrage, ob z.B. ein bestimmter Schalter gesetzt ist, und Verzweigung zur nächsten Abfrage

(02) Steht OKIN noch auf Null, dann wird die entsprechende Meldung ausgegeben und die Generierung beendet.

MNOTE Meldungen absetzen

Funktion

Die MNOTE-Anweisung erzeugt eine durch variable Parameter veränderliche Meldung mit einer Zeilennummer im Übersetzungsprotokoll.

Format

Name	Operation	Operanden
[.sym]	MNOTE	$[\left\{\begin{matrix}*\\ \\ n\end{matrix}\right\},]$'text'

.sym	Folgesymbol
n	Fehlercode, positiver arithmetischer Makroausdruck; $0 \leq n \leq 255$
text	Text der Meldung, maximal 256 Zeichen

Beschreibung

Abhängig vom Fehlercode in der MNOTE-Anweisung können im Übersetzungsprotokoll eigene Warnungen oder Fehlermeldungen ausgegeben werden. Eine Fehlermeldung, die aufgrund einer MNOTE ausgegeben wurde, wird analog den Assembler-Flags behandelt und das Fehlergewicht entsprechend erhöht.

Entsprechend der Fehler-Referenzliste kann eine Mnote-Referenzliste erstellt werden (siehe ASSEMBH, Benutzerhandbuch [1]).

* Wird als Fehlercode ein Stern angegeben, dann erscheint text als Kommentarzeile im generierten Code, auch wenn die Anweisung PRINT NOGEN gesetzt wurde. Die Meldung ist reiner Kommentartext, es werden keine Fehlercodes mitgezählt. Eine Ausgabe im MNOTE-XREF erfolgt nicht.

n Der Fehlercode der MNOTE-Anweisung und die Fehlerklassen der Assembler-Flags hängen wie folgt zusammen:

Warnung	$n = 0$
Fehler	$1 \leq n \leq 150$
schwerer Fehler	$151 \leq n \leq 254$
Abbruchfehler	$n = 255$

Wird n weggelassen oder ist $n > 255$, nimmt der Assembler $n = 0$ an. Bei $n > 255$ wird zusätzlich eine Warnung ausgegeben.

Ist n nicht ganzzahlig, dann gilt die MNOTE-Anweisung als fehlerhafte Modellanweisung.

Programmierhinweise

1. Variable Parameter im Operandeneintrag der MNOTE-Anweisung werden durch ihren aktuellen Wert ersetzt. Durch MNOTE *,... können in diesem Fall Kommentare generiert werden.
2. Durch eine MNOTE-Anweisung mit dem Fehlercode n = 255 kann die Übersetzung vorzeitig abgebrochen werden, wenn bei der Option-Angabe die entsprechende Abbruchbedingung gesetzt ist (siehe ASSEMBH, Benutzerhandbuch [1]).

MTRAC Macro trace

Funktion

Die MTRAC-Anweisung dient der Überwachung von bedingten Verzweigungen und der Kontrolle der Inhalte von SET-Parametern.

Format

Name	Operation	Operanden
[.sym]	MTRAC	

.sym Folgesymbol

Beschreibung

Die MTRAC-Anweisung bewirkt, daß nach ihrem Auftreten folgende Makroanweisungen im Übersetzungsprotokoll ausgedruckt werden:

Die MTRAC-, NTRAC-, ANOP-, LCLx- und GBLx-Anweisung;
Diese Anweisungen werden ohne weitere Angaben protokolliert.

Die AIF- und AGO-Anweisung;
Bei diesen beiden Anweisungen wird im Übersetzungsprotokoll durch ein "Y" oder "N" angezeigt, ob verzweigt wurde oder nicht.

einfaches AIF	erweitertes AIF	einfaches AGO	computed AGO
Y N	Y: <n> N	Y	Y: <n>

Y es wurde verzweigt
N es wurde nicht verzweigt
n ist die Nummer des logischen Ausdrucks, der zur Verzweigung führt

Die SETA-, SETB-, SETC- und die ACTR-Anweisung;
Hier wird im Übersetzungsprotokoll der augenblickliche Wert der SET-Parameter, bzw. des ACTR-Operanden ausgedruckt.

- SETA-Parameter und ACTR-Operand werden als Dezimalzahlen mit führenden Nullen dargestellt. Negative Werte der SETA-Parameter werden mit Vorzeichen ausgedruckt. Zahlen mit mehr als 8 Stellen werden rechts abgeschnitten.
- Der Wert eines SETB-Parameters erscheint als einzelnes Zeichen:

  T ≙ wahr (true), bzw. 1
  F ≙ falsch (false), bzw. 0
- SETC-Parameter oder symbolische Parameter in SETC-Anweisungen werden dargestellt als Zeichenfolge von maximal 8 Zeichen.

  Für einen SETC-Parameter, der auf den Anfangswert zurückgesetzt wurde, steht die Zeichenfolge 'NULL␣␣␣␣'.

  Ist der Wert des SETC-Parameters länger als 8 Zeichen, werden nur die ersten 7 Zeichen und ein Stern (*) ausgedruckt.

Programmierhinweise

1. Wenn für die Steuerung des Protokollinhalts PRINT NOGEN angegeben wurde, dann wird auch keine MTRAC-Information ausgegeben.
2. Steht die MTRAC-Anweisung im Quellprogramm oder in einem äußeren Makro, dann muß sie durchlaufen werden, damit sie für die folgenden inneren Makroaufrufe gültig ist.
3. Die MTRAC-Eigenschaft wird durch die Aufrufhierarchie, abhängig von der Makroverschachtelungstiefe, nach unten weitervererbt (siehe Beispiel bei NTRAC).

NTRAC Macro trace beenden

Funktion

Die NTRAC-Anweisung setzt die MTRAC-Funktion außer Kraft.

Format

Name	Operation	Operanden
[.sym]	NTRAC	

.sym Folgesymbol

Programmierhinweise

1. Ist keine MTRAC-Anweisung angegeben, sind alle verarbeiteten NTRAC-Anweisungen wirkungslos.
2. NTRAC setzt die MTRAC-Anweisung in der entsprechenden Makroverschachtelungsstufe zurück. NTRAC gilt auch für alle folgenden inneren Makroaufrufe.

Beispiel

Das Beispiel zeigt die Vererbung der MTRAC-, bzw. NTRAC-Eigensschaft in einem Quellprogramm, das mehrere Makroaufrufe ineinander verschachtelt enthält.

Das Quellprogramm ruft den Makro MAC1 auf, dieser den Makro MAC2, und dieser den Makro MAC3.

```
CSECT
 .
 .
MTRAC                                  ┐
 .                                     ├ MTRAC gilt
 .       MACRO                         ┘
MAC1 ──> MAC1
          .                            ┐ in MAC1 keine Angabe von MTRAC oder NTRAC;
          .       MACRO                ┘ d.h., MTRAC gilt weiter
         MAC2 ──> MAC2
                   .                   ┐ MAC2 enthält NTRAC; d.h., MTRAC
                   .                   ├ gilt nur bis zur NTRAC-Anweisung,
                  NTRAC                ┘ ab hier gilt NTRAC
                   .
                   .       MACRO
                  MAC3 ──> MAC3        ┐ in MAC3 keine Angabe von MTRAC oder NTRAC;
                            .          ┘ d.h., NTRAC gilt weiterhin
                            .
                       <──  .
                   .       MEND        ┐ Rückkehr in MAC2; wie zuletzt in
                   .                   ┘ dieser Stufe: NTRAC gilt
               <──  .
          .       MEND                 ┐ Rückkehr in MAC1; wie bisher in
          .                            ┘ dieser Stufe: MTRAC gilt
     <──  .
 .       MEND                          ┐ Rückkehr ins Quellprogramm;
 .                                     ┘ wie bisher: MTRAC gilt
END
```

SETA SETA-Parameter setzen

Funktion

Die SETA-Anweisung weist einem SETA-Parameter den arithmetischen Wert aus dem Operandeneintrag zu.

Format

Name	Operation	Operanden
[&par] {&par(d)}	SET[A]	[arexp] {&par_c}

&par	SETA-Parameter
&par(d)	indizierter SETA-Parameter;
d	Dimension; arithmetischer Makroausdruck
arexp	arithmetischer Makroausdruck
&par_c	SETC-Parameter

Beschreibung

&par, bzw. &par(d)

lokaler oder globaler SETA-Parameter, der in einer LCLA- oder GBLA-Anweisung definiert wurde.

Wurde der SETA-Parameter nicht vor der SETA-Anweisung definiert, dann interpretiert der Assembler ihn als impliziert vereinbarten lokalen SETA-Parameter und weist als Anfangswert den Wert von arexp zu. In diesem Fall muß man den mnemotechnischen Operationscode SETA verwenden.

Mehrere aufeinander folgende SETA-Anweisungen mit dem gleichen SETA-Parameter im Namenseintrag weisen diesem jeweils einen neuen Wert zu.

arexp Der Wert des Ausdrucks im Operandeneintrag wird nach den Regeln für arithmetische Makroausdrücke berechnet und dann dem SETA-Parameter zugewiesen.

Dieser Wert wird an Stelle des SETA-Parameters eingesetzt, wenn der Parameter in einem arithmetischen Ausdruck verwendet wird.

&par_c Ein SETC-Parameter als Operandeneintrag ist nur zulässig, wenn er einen arithmetischen Wert hat.

Eine leere Zeichenfolge wird in eine 0 konvertiert.

Programmierhinweis

Wird der SETA-Parameter in einem nicht-arithmetischen Makroausdruck verwendet, dann wird der Wert von arexp in eine ganze Zahl ohne führende Nullen umgewandelt.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------
* Makrodefinition
            | MACRO     |
&NAME       | SETAEX    | &PART,&PARF
            | LCLA      | &P1,&P2,&P3,&P4
.*          |           |
&P1         | SETA      | 10                  (01)
&P2         | SETA      | 12                  (02)
&P3         | SETA      | &P1-&P2             (03)
&P4         | SETA      | &P1+&P3             (04)
.*          |           |
&NAME       | ST        | 2,SAVAREA
            | L         | 2,&PARF&P3          (05)
            | ST        | 2,&PART&P4          (06)
            | L         | 2,SAVAREA
            | MEND      |

* Makroaufruf
BEGIN       | SETAEX    | FIELDX,FIELDY

* Generierte Instruktionen
            | .         |
            | .         |
BEGIN       | ST        | 2,SAVAREA
            | L         | 2,FIELDY2
            | ST        | 2,FIELDX8
            | L         | 2,SAVAREA
            | .         |
            | .         |
```

(01) und (02) ordnen den SETA-Parametern &P1 und &P2 die arithmetischen Werte 10 und 12 zu.

(03) und (04) Verwendung der SETA-Parameter in arithmetischen Makroausdrücken, &P3 und &P4 erhalten die Werte -2, bzw +8.

(05) und (06) Verwendung der SETA-Paramter in nicht-arithmetischen Makroausdrükken, deshalb werden &P3, bzw. &P4 hier durch 2, bzw. 8 ersetzt.

SETB SETB-Parameter setzen

Funktion

Die SETB-Anweisung weist einem SETB-Parameter den logischen Wert 1 oder 0 (wahr oder falsch) zu.

Format

Name	Operation	Operanden
[&par] {&par(d)}	SET[B]	{0 1 (0) (1) (logexp) &par_c}

&par SETB-Parameter
&par(d) indizierter SETB-Parameter
d Dimension; arithmetischer Makroausdruck
logexp logischer Ausdruck
&par_c SETC-Parameter

Beschreibung

&par, bzw. &par(d)
: lokaler oder globaler SETB-Parameter, der in einer LCLB- oder GBLB-Anweisung definiert wurde.

  Wurde der SETB-Parameter nicht vor der SETB-Anweisung definiert, dann interpretiert ihn der Assembler als implizit vereinbarten lokalen SETB-Parameter und weist als Anfangswert 0 oder 1, je nach Wert des Operandeneintrags, zu. In diesem Fall muß der mnemotechnische Operationscode SETB verwendet werden.

  Mehrere aufeinanderfolgende SETB-Anweisungen mit dem gleichen SETB-Parameter im Namenseintrag weisen diesem jeweils einen neuen Wert zu.

0, 1, (0) oder (1)
: Diese Werte werden dem SETB-Parameter zugewiesen und an seiner Stelle eingesetzt, wenn er in einem logischen Ausdruck verwendet wird.

logexp Der logische Wert des Ausdrucks im Operandeneintrag wird nach den Regeln für logische Ausdrücke berechnet und dann dem SETB-Parameter zugewiesen.

Dieser Wert wird an Stelle des SETB-Parameters eingesetzt, wenn dieser in einem logischen Ausdruck verwendet wird.

&par_c Ein SETC-Parameter als Operandeneintrag ist nur zugelassen, wenn er einen Wert von 0 oder 1 hat.

Eine leere Zeichenfolge wird in eine 0 konvertiert.

Programmierhinweis

Wird ein SETB-Parameter in einem arithmetischen Makroausdruck verwendet, dann werden die logischen Werte 1 (wahr) oder 0 (falsch) in die entsprechenden arithmetischen Werte +1 oder +0 umgewandelt.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------
* Makrodefinition
            | MACRO     |
&NAME       | SETBEX    | &P1,&P2
            | LCLB      | &B1,&B2
            | .         |
            | .         |
&B1         | SETB      | (L'&P1 EQ 4)        (01)
&B2         | SETB      | (S'&P2 EQ 0)        (02)
            | .         |
            | .         |
            | MEND      |

* Makroaufruf
BEGIN       | SETBEX    | FIELDA,FIELDB

* Definitionen im Quellprogramm
FIELDA      | DC        | F'01'
FIELDB      | DC        | DS3'12'
            | .         |
            | .         |
```

(01) Der logische Ausdruck ist wahr, daher wird &B1 der Wert 1 zugewiesen.
(02) Der logische Ausdruck ist falsch, daher wird &B2 der Wert 0 zugewiesen.

SETC SETC-Parameter setzen

Funktion

Die SETC-Anweisung weist einem SETC-Parameter oder einem symbolischen Parameter den Wert des Zeichenausdrucks im Operandeneintrag zu.

Format

Name	Operation	Operanden
[&par &par(d) &spar &SYSLIST(n)]	SET[C]	charexp

&par	SETC-Parameter
&par(d)	indizierter SETC-Parameter
d	Dimension; arithmetischer Makroausdruck
&spar	symbolischer Parameter
&SYSLIST(n)	variabler Systemparameter
charexp	Zeichenausdruck

Beschreibung

&par, bzw. &par(d)
: lokaler oder globaler SETC-Parameter, der in einer LCLC- oder GBLC-Anweisung definiert wurde.

  Wurde der SETC-Parameter nicht vor der SETC-Anweisung definiert, dann interpretiert der Assembler ihn als impliziten lokalen SETC-Parameter und weist als Anfangswert den Wert von charexp zu. In diesem Fall muß der mnemotechnische Operationscode SETC verwendet werden.

  Mehrere aufeinander folgende SETC-Anweisungen mit dem gleichen SETC-Parameter im Namenseintrag weisen diesem jeweils einen neuen Wert zu.

&spar, bzw. &SYSLIST(n)
: symbolischer Parameter, dem der Wert im Operandeneintrag zugewiesen werden soll. Bei Verwendung von &SYSLIST(n) darf n nur den symbolischen Parameter selbst bezeichnen. Einem Element einer Operandenunterliste kann mit SETC kein neuer Wert zugewiesen werden.

charexp
: Der Wert des Zeichenausdrucks wird an Stelle des SETC-Parameters eingesetzt, wenn dieser in einer Instruktion verwendet wird.

Programmierhinweis

Bei der SETC-Bearbeitung wird der Wert des Operanden immer als String interpretiert und nicht in Unterlisten zerlegt; z.B. bei &PAR SETC '(1,2)' ist der Wert von &PAR(1) = (1,2) und nicht 1.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------
* Makrodefinition
            | MACRO     |
&NAME       | SETCEX    | &IN,&OUT
            | LCLC      | &PRE
*           |           |
&PRE        | SETC      | '&IN'(1,5)          (01)
*           |           |
&NAME       | ST        | 2,SAVAREA
            | L         | 2,&PRE&OUT          (02)
            | ST        | 2,&IN
            | L         | 2,SAVAREA
            | MEND      |

* Makroaufruf
BEGIN       | SETCEX    | FIELDA,B

* Generierte Instruktionen
            | .         |
            | .         |
BEGIN       | ST        | 2,SAVAREA
            | L         | 2,FIELDB
            | ST        | 2,FIELDA
            | L         | 2,SAVAREA
            | .         |
            | .         |
```

(01) Die SETC-Anweisung ordnet &PRE den Wert der Teilzeichenfolge (FIELD) zu.
(02) &PRE wird ersetzt durch FIELD und &OUT wird ersetzt durch B, d.h., &PRE&OUT erhält den Wert FIELDB

8 Makrosprachelemente im Assembler-Quellprogrammtext

Verschiedene Elemente der Makrosprache können **außerhalb** von Makrodefinitionen im Text des Assembler-Quellprogramms verwendet werden. In diesem Fall ist das Assembler-Quellprogramm wie eine große Makrodefinition zu betrachten, nur ohne MACRO-, MEND- und Musteranweisung.

Im Übersetzungsprotokoll stehen die Instruktionen, die mit Hilfe der Makrosprache generiert wurden, jeweils direkt hinter den entsprechenden Makro-Instruktionen und sind durch ein Pluszeichen (+) in der Spalte für die Makrostufe gekennzeichnet.

Die Makrosprachelemente, die im Assembler-Quellprogramm eingesetzt werden können, sind im folgenden aufgeführt.

- **Makroanweisungen**

	Verwendung außerhalb von Makrodefinitionen
ACTR	ja
AIF	ja
AGO	ja
ANOP	ja
GBLx	ja
LCLx	ja
MACRO	nein
MEND	nein
MEXIT	nein
MNOTE	ja
MTRAC	ja
NTRAC	ja
SETA	ja
SETB	ja
SETC	ja

Programmierhinweise

1. Durch AIF oder AGO übersprungene Instruktionen werden nicht gelesen und daher auch nicht übersetzt. In übersprungenen Instruktionen definierte Namen gelten daher für den Assembler als nicht definiert. Auf ihre Merkmale kann aber auf Grund der Lookahead-Funktion des ASSEMBH zugegriffen werden (siehe ASSEMBH, Benutzerhandbuch [1]).
2. Instruktionen, die in einer Schleife mehrfach durchlaufen werden, werden auch mehrfach hintereinander eingelesen und übersetzt.
3. Eine Überschreitung des ACTR-Zählers (siehe 7.2, ACTR-Anweisung) führt hier zu einem Abbruch der Übersetzung.

- **Variable Parameter**

	Verwendung außerhalb von Makrodefinitionen
Symbolische Par.	nein
SET-Parameter	ja, wenn sie vor ihrer ersten Verwendung definiert sind *Ausnahme:* implizit vereinbarte lokale SET-Parameter müssen nicht vorher definiert werden.
Variable Systempar.	
&SYSDATE	ja
&SYSECT	nein
&SYSLIST	nein
&SYSMOD	ja
&SYSNDX	nein
&SYSPARM	ja
&SYSTEM	ja
&SYSTIME	ja
&SYSTSEC	nein
&SYSVERM	nein
&SYSVERS	ja

Programmierhinweis

Auch im Assembler-Quellprogramm können variable Parameter verwendet werden, um den Namens-, Operations- und Operandeneintrag von Assembler-Instruktionen zu generieren.

- **Folgesymbole**

	Verwendung außerhalb von Makrodefinitionen
Standardformat	ja, das Folgesymbol wird protokolliert
generiertes Format	ja, im Operandeneintrag

Programmierhinweis

Folgesymbole im Operandeneintrag von AIF- und AGO-Anweisungen müssen auch im Namenseintrag einer Instruktion im Quellprogramm definiert sein.

- **Bezugnahmen auf Merkmale**

	Verwendung außerhalb von Makrodefinitionen
Typenmerkmal	ja, in Makroanweisungen
Längenmerkmal	ja
Skalenfaktormerkmal	ja, in Makroanweisungen
Ganzzahligkeitsmerk.	ja, in Makroanweisungen
Zählermerkmal	ja, in Makroanweisungen für SET-Parameter und variable Systemparameter
Anzahlmerkmal	nein
Definitionsmerkmal	ja, in Makroanweisungen

9 Strukturierte Programmierung mit ASSEMBH

9.1 Einführung

Die Strukturierte Programmierung wird vom ASSEMBH-BC nicht unterstützt !

Die strukturierte Programmierung enthält Regeln für die Konstruktion von Programmen. Das Ziel ist ein klarer, gut verständlicher Programm-Steuerfluß.

Jedes Programm hat einen statischen und einen dynamischen Aspekt. Der statische Aspekt drückt sich aus in der Niederschrift des Programms, dem Quellprogramm-Text. Der dynamische Aspekt besteht aus der Folge von Aktionen, die das Programm im Rechner auslöst. Im Text eines Quellprogramms liegen Instruktionen vor, die festlegen, welche Aktion als nächste ausgeführt werden soll. Diese Instruktionen legen den Steuerfluß des Programms fest.

Die strukturierte Programmierung geht davon aus, daß der Steuerfluß eines Programms nur aus bestimmten Grundmustern bestehen darf. Hierfür wird auf den folgenden Grundkonzepten aufgebaut:

- **Blockkonzept**

  Das Blockkonzept besteht in der Forderung, daß ein Programm nur durch Aneinanderreihen oder Schachteln von Strukturblöcken gebildet werden darf, die jeweils genau einen Eingang und einen Ausgang besitzen. Der Steuerfluß eines Programms darf nur aus einigen wenigen Grundmustern dieser Art mit einer einfachen inneren Struktur aufgebaut werden.

- **Prozedurkonzept**

  Im Prozedurkonzept wird der Kontrollfluß zwischen Hauptprogrammen und Unterprogrammen geregelt. Eine Prozedur im Sinne der strukturierten Programmierung ist eine Programmeinheit mit einem Eingang und einem Ausgang, die unter einem Namen, ggf. mit aktuellen Parameterwerten, aufgerufen werden kann. Eine Prozedur ist aus Strukturblöcken aufgebaut.

- **Datenkonzept**

  Das Datenkonzept verhilft dem Anwender zur rationellen Speichernutzung und nimmt ihm die Speicherverwaltung ab, die für die Erzeugung reentrant-fähiger Prozeduren notwendig ist. Das Datenkonzept regelt den Gültigkeitsbereich aller Daten und sorgt für Übersichtlichkeit. Es verlangt, daß jede Prozedur über alle verwendeten Daten Auskunft gibt.

Der ASSEMBH stellt dem Anwender Hilfsmittel zur Verfügung, um Assembler-Programme den Anforderungen der strukturierten Programmierung anzupassen. Dies geschieht mit Hilfe eines Satzes von vordefinierten Makros, mit denen sich die genannten Grundkonzepte realisieren lassen.

Beim Ablauf eines Programms mit vordefinierten Makros arbeitet der ASSEMBH mit dynamischer Speicherverwaltung. Für Speicherzugriff und -anforderung, automatische Registersicherung und Parameterverwaltung sorgt hierbei das Laufzeitsystem des ASSEMBH. Das Laufzeitsystem wird realisiert durch ein Modulpaket, das in der zugehörigen Modulbibliothek vorliegt.

Neben dem Makrosatz und dem Laufzeitsystem gehören zur Strukturierten Programmierung mit ASSEMBH die Dienstprogramme zur Fehlersuche und Listenaufbereitung.

Der folgende Text beschreibt das Block-, das Prozedur- und das Datenkonzept im ASSEMBH, sowie die vordefinierten Makros und ihre Verwendung. Die Handhabung der Dienstprogramme und der Makrobibliothek, sowie das Einbinden des Laufzeitsystems sind im ASSEMBH, Benutzerhandbuch [1] beschrieben.

Verwendung der Strukturierten Programmierung für den Programmentwurf

Die Grundkonzepte der strukturierten Programmierung schaffen die Voraussetzung für ein schrittweise verfeinerndes Vorgehen beim Programmentwurf.

Für die Formulierung der Ablauflogik eines Programms bietet sich die Verwendung von Pseudocode an. Pseudocode besteht aus formalisierten Teilen und aus Texten in natürlicher Sprache. Dabei können für die formalisierten Teile die vordefinierten Makros des ASSEMBH verwendet werden. Der Steuerfluß eines Programms wird mit den vordefinierten Makros formuliert, die Funktion der einzelnen Blöcke in natürlicher Sprache umschrieben. Im nächsten Schritt kann die Funktion der Einzelblöcke detailliert ausgearbeitet oder einzelne Blöcke als eigene Prozeduren ausgelagert werden.

Beispiel

Das Beispiel zeigt eine Entscheidung (siehe 9.2.2, Auswahlstrukturblöcke), in der bisher nur die Makros angegeben sind, die den Strukturblock darstellen.

```
@IF
Bedingung
@THEN
Operationen, falls Bedingung zutrifft
@ELSE
Operationen, falls Bedingung nicht zutrifft
@BEND
```

Die Dienstprogramme des ASSEMBH werten die vordefinierten Makros zu übersichtlichen Darstellungen aus, die auf dem Drucker ausgegeben werden können, und prüfen den Entwurf auf Strukturfehler. Da die zwischen den Makros stehenden Texte von den Dienstprogrammen nicht bearbeitet werden, läßt sich mit diesem Pseudocode bereits der Programmentwurf strukturiert darstellen, ohne daß Codierung in Assembler vorliegt.

Soll dann aus dem in Pseudocode vorliegenden Entwurf ein Assembler-Programm erarbeitet werden, so genügt es, die Funktionen der einzelnen Blöcke in Assembler-Instruktionen auszudrücken. Dabei empfiehlt es sich, die Text-Teile des Pseudocode als Kommentare im Quellprogramm-Text zu belassen. Die vordefinierten Makros bilden nach wie vor den Steuerfluß des implementierten Moduls.

Der ASSEMBH setzt die vordefinierten Makros in Assembler-Instruktionen um und übersetzt sie zusammen mit den übrigen Instruktionen in den entsprechenden Objektmodul.

9.2 Blockkonzept

Das Blockkonzept setzt voraus, daß die Beschreibung des Programmablaufs und der Aufbau einer Prozedur durch wenige Grundmuster, die "Strukturblöcke" erfolgt. Eine Prozedur darf nur aus aneinandergereihten oder geschachtelten Strukturblöcken bestehen.

Jeder Strukturblock hat einen einfachen Steuerfluß, der die Reihenfolge der in ihm enthaltenen Arbeitsschritte festlegt. Grundsätzlich gibt es nur drei Formen des Steuerflusses in einem Strukturblock:

- die Sequenz (mehrere Schritte folgen aufeinander); Abschnitt 9.2.1
- die Auswahl (einer von mehreren möglichen Schritten wird ausgeführt); Abschnitt 9.2.2
- die Schleife (ein Schritt wird mehrmals wiederholt); Abschnitt 9.2.3

Dadurch hat der Steuerfluß in einem Strukturblock nur je einen Eingang und einen Ausgang.

Mehrere Strukturblöcke können aneinandergereiht werden und ergeben somit wieder eine Sequenz, also einen übergeordneten Strukturblock. Außerdem kann man Strukturblöcke schachteln; d.h. ein Schritt in einer Sequenz, eine Alternative eines Auswahlstrukturblocks oder der wiederholte Teil einer Schleife kann selbst aus einem Strukturblock bestehen.

In den Strukturblöcken Entscheidung, Schleife mit Vorabprüfung, Schleife mit freier Endebedingung und Zählschleife mit freier Endebedingung ist für die Makroaufrufe @IF, @WHIL und @WHEN die Angabe einer Bedingung erforderlich.
Es gibt:

- einfache Bedingungen; Abschnitt 9.2.4
- zusammengesetzte Bedingungen; Abschnitt 9.2.5

Für die grafische Darstellung der Strukturblöcke werden im allgemeinen Nassi-Shneiderman-Diagramme (Struktogramme) verwendet. Der ASSEMBH bietet dem Anwender einen Satz vordefinierter Makros, mit denen jedes Struktogramm 1:1 codiert werden kann.

Alle Verzweigungen in einem Programm werden mit Hilfe dieser Makros realisiert, deshalb sollten Sprungbefehle (B, BAL, BALR, ...) grundsätzlich nicht verwendet werden.

9.2.1 Sequenz

Der Sequenz-Strukturblock faßt Programmteile zu einer logischen Einheit zusammen. Er ist beliebig einsetzbar, dient vor allem der übersichtlichen Gliederung und ermöglicht die Vergabe von Namen an Folgen von Instruktionen und Strukturblöcken.

Der Sequenz-Strukturblock ist vorgeschrieben für die Abgrenzung der Unterblöcke der Fallunterscheidung durch Nummer (@CASE).

Darstellung nach Nassi-Shneiderman

Format des Strukturblocks

Name	Operation	Operanden
[name]	@BEGI	
	{ Instruktion / Strukturblock } [,...]	
[name]	@BEND	

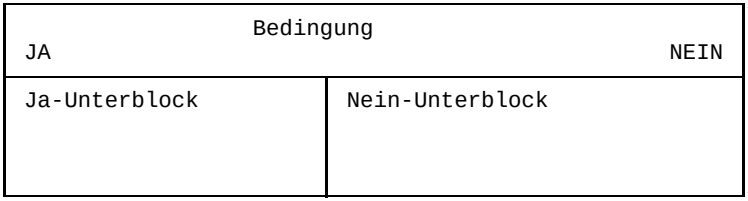
9.2.2 Auswahlstrukturblöcke

Ein Auswahlstrukturblock besteht aus einer Reihe von Unterblöcken und einem Kriterium, das aus den Unterblöcken genau einen auswählt.

9.2.2.1 Entscheidung

Bei der Entscheidung wird von zwei Unterblöcken einer ausgewählt. Das Auswahlkriterium besteht in einer Bedingung, die sich zusammensetzt aus einem Bedingungssymbol und einem Anzeige-setzenden Assemblerbefehl (siehe 9.2.2, Bedingungen).

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@IF	cond_sym [Anzeige-setzender Assemblerbefehl]
[name]	@THEN	Ja-Unterblock
[[name]	@ELSE	Nein-Unterblock]
[name]	@BEND	

cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol; gibt an, worauf die Bedingung abgefragt werden soll.

Unterblock

Folge von Instruktionen oder weiteren geschachtelten Strukturblöcken

Beschreibung

Der Ja-Unterblock wird durch den @THEN-Makro eingeleitet, der Nein-Unterblock durch @ELSE. @BEND schließt den Strukturblock ab.

Ist die Bedingung (siehe 9.2.4 und 9.2.5) wahr, werden die Instruktionen, bzw. Strukturblöcke im Ja-Unterblock ausgeführt, andernfalls der Nein-Unterblock. Fehlt der Nein-Unterblock wird bei nicht zutreffender Bedingung die Instruktion ausgeführt, die auf den @BEND-Makro folgt.

Beispiel

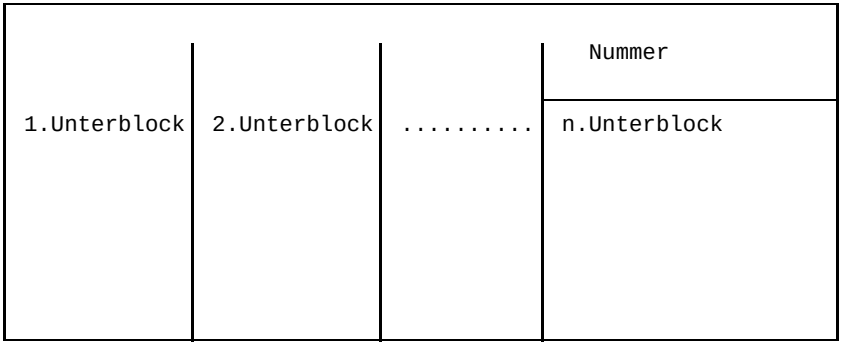
```
Name        | Operation | Operanden
------------+-----------+-----------------
            | @IF       | LE
            | CR        | R1,R2
            | @THEN     |
            | MVI       | FIELD,TRUE
            | @ELSE     |
            | MVI       | FIELD,FALSE
            | @BEND     |
```

Wenn der Inhalt von Register 1 kleiner oder gleich dem von Register 2 ist, wird der Ja-Unterblock ausgeführt, sonst der Nein-Unterblock.

9.2.2.2 Fallunterscheidung durch Nummer

Bei der Fallunterscheidung durch Nummer wird von mehreren Unterblöcken einer ausgewählt. Das Auswahlkriterium ist die Nummer des entsprechenden Unterblocks. Diese Nummer wird im @CASE-Register angegeben.

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@CASE	(reg)
[name]	@BEGI	
	1. Unterblock	
[name]	@BEND	
	.	
	.	
	.	
[name]	@BEGI	
	n. Unterblock	
[name]	@BEND	
[name]	@BEND	

reg @CASE-Register, enthält die Nummer des auszuführenden Unterblocks
Unterblock Folge von Instruktionen oder weiteren geschachtelten Strukturblöcken

Beschreibung

Das @CASE-Register muß vor der Ausführung des Strukturblocks mit der Nummer des entsprechenden Unterblocks geladen werden. Für den Strukturblock sind maximal 90 Unterblöcke möglich.

Die Verwendung von Register 0 als @CASE-Register ist unzulässig.

Der Inhalt des @CASE-Registers wird bei der Ausführung des @CASE-Makro verändert.

Ist der Inhalt des @CASE-Registers kleiner als 1 oder größer als die Zahl der definierten Unterblöcke,
- wird der letzte Unterblock ausgeführt, falls im Prozedurkopf CHECK=ON gesetzt ist (siehe 10.2, @ENTR);
- treten Programmfehler auf, falls im Prozedurkopf CHECK=OFF gesetzt ist.

Die Unterblöcke in einer Fallunterscheidung durch Nummer müssen mit einer Sequenz (@BEGI, @BEND) abgegrenzt werden.

Beispiel

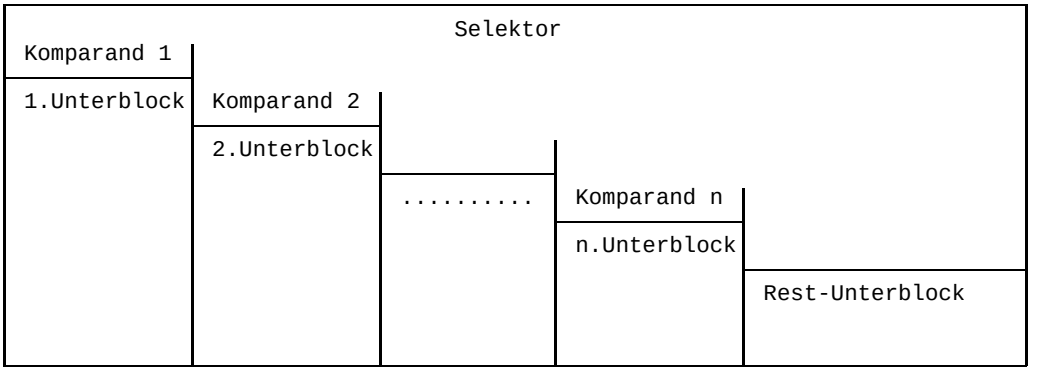
```
Name        Operation   Operanden
            L           R6,T2
            @CASE       (R6)
            @BEGI
            .                       } 1. Unterblock
            .
            @BEND
            @BEGI
            .                       } 2. Unterblock
            .
            @BEND
            .                       } weitere Unterblöcke
            .                       } möglich
            @BEND
            .
            .
T1          DC          F'1'
T2          DC          F'2'
            .
            .
```

Im Beispiel wird Register 6 für die @CASE-Verzweigung verwendet. Der Inhalt von Register 6 ist hier 2, deshalb wird der zweite Unterblock ausgeführt.

9.2.2.3 Fallunterscheidung durch Vergleich

Bei der Fallunterscheidung durch Vergleich wird von mehreren Unterblöcken einer ausgewählt. Dies geschieht auf Grund des Vergleichs von Selektor und Komparanden. Der Selektor wird im Kopf des Strukturblocks angegeben, die Komparanden im Kopf der Unterblöcke.

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@CAS2	{name / literal / (reg)}[,COMP=instr]
[name]	@OF	{name / literal / val}[,...][,COMP=instr]
	1. Unterblock	
	.	
	.	
	.	
[[name]	@OFRE	
	Rest-Unterblock]	
[name]	@BEND	

name, bzw. literal oder (reg) im Operandeneintrag von @CAS2
: gibt den Namen des Feldes oder das Register an, das den Selektor enthält, bzw. in Form eines Literals, den Selektor selbst.

name, bzw. literal oder val im Operandeneintrag eines @OF
: gibt den Namen des Feldes an, das den entsprechenden Komparanden enthält, bzw. in Form eines Literals oder eines selbstdefinierenden Wertes, den Komparanden selbst.

Unterblock
: Folge von Instruktionen oder weiteren geschachtelten Strukturblöcken

instr
: Anzeige-setzender Assemblerbefehl

Beschreibung

Der aktuelle Wert des Selektors im @CAS2-Makro wird mit den Komparanden in den @OF-Makros verglichen. Der Vergleich erfolgt in der angegebenen Reihenfolge der Komparanden, von oben nach unten und von links nach rechts. Bei der ersten Gleichheit von Selektor und Komparand wird der zugehörige Unterblock abgearbeitet und der Strukturblock anschließend verlassen.

Liegt kein passender Komparand vor, wird der Restunterblock bearbeitet. Bei dessen Fehlen wird der Strukturblock verlassen und die Instruktion bearbeitet, die auf den @BEND-Makro folgt.

Das Ende eines Unterblocks ist gekennzeichnet durch den nächsten @OF, @OFRE oder den @BEND-Makro, der den Strukturblock abschließt.

Standardmäßig erfolgt der Vergleich mit dem CLC-Befehl, bei Registern mit dem C-Befehl. Mit dem Operanden COMP= kann eine abweichende Vergleichsart gewählt werden. Dabei gilt der Operand im @CAS2 für den ganzen Strukturblock, in den @OF-Makros nur für den jeweiligen Unterblock.

Der Anwender muß selbst dafür sorgen, daß die generierten Assembler-Befehle

```
Vergleichsbefehl      Selektor,Komparand
```

korrekt sind, das heißt u.a., daß implizite oder explizite Längenangaben und die Ausrichtung der Operanden richtig sind.

Bei den Vergleichsbefehlen wird der Unterblock bei Gleichheit (EQ) ausgeführt; bei COMP=TM, falls die der Maske entsprechenden Bits im Selektor alle gesetzt sind (ON); bei COMP=TRT, falls im Komparanden (Umsetzungstafel) ein Funktionsbyte ungleich Null (NZ) gefunden wurde.

Die Anzahl der Unterblöcke im Strukturblock und die Anzahl der Komparanden in einem @OF sind nicht begrenzt.

Für die Auswahl des richtigen Unterblocks werden alle Vergleiche, die vor dem gesuchten Unterblock stehen, ausgeführt. Es empfiehlt sich daher, die Unterblöcke und Komparanden nach zu erwartender Häufigkeit zu ordnen.

Beispiel

```
Name      | Operation | Operanden
----------+-----------+-------------------
BLOC      | @CAS2     | KEY,COMP=CLI        (01)
          | @OF       | 5                   (02)
          | .         |
          | .         |
          | @OF       | 6,8                 (03)
          | .         |
          | .         |
          | @OF       | END,COMP=CLC        (04)
          | .         |
          | .         |
          | @OF       | X'80',COMP=TM       (05)
          | .         |
          | .         |
          | @BEND     |
```

(01) Der Selektor liegt vor im Feld KEY, der Vergleichsbefehl für den Strukturblock ist CLI.

(02) Komparand für den ersten Unterblock ist der selbstdefinierende Wert 5. D.h., der erste Unterblock wird ausgeführt, wenn der aktuelle Wert von KEY gleich 5 ist.

(03) Für den zweiten Unterblock sind als Komparanden die selbstdefinierenden Werte 6 und 8 angegeben. D.h., er wird ausgeführt, wenn der Wert von KEY gleich 6 oder gleich 8 ist.

(04) Änderung des Vergleichsbefehls für diesen Unterblock; er wird bearbeitet, falls der Wert von KEY gleich dem Wert von END ist.

(05) Für den letzten Unterblock wird erneut ein anderer Vergleichsbefehl angegeben. Er wird ausgeführt, falls die der Maske entsprechenden Bits im Selektor alle gesetzt sind.

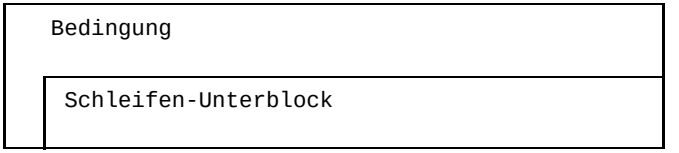
9.2.3 Schleifen

Eine Schleife besteht aus einem Schleifen-Unterblock und einer Bedingung, die angibt, wie oft, bzw. wie lange dieser wiederholt werden soll.

9.2.3.1 Schleife mit Vorabprüfung

Bei einer Schleife mit Vorabprüfung wird jeweils vor Ausführung des Schleifen-Unterblocks die Bedingung geprüft.

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@WHIL	cond_sym [Anzeige-setzender Assemblerbefehl]
[name]	@DO	Schleifen-Unterblock
[name]	@BEND	

cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol; gibt an, worauf die Bedingung abgefragt werden soll.

Schleifen-Unterblock
Folge von Instruktionen oder weiteren geschachtelten Strukturblöcken

Beschreibung

Der Schleifenunterblock wird ausgeführt, falls die Bedingung (siehe 9.2.4 und 9.2.5) erfüllt ist. Ist die Bedingung nicht erfüllt, wird die Schleife vor dem Schleifenunterblock verlassen.

Wenn bei der ersten Prüfung die Bedingung nicht erfüllt ist, wird der Schleifenunterblock nicht durchlaufen.

Beispiel

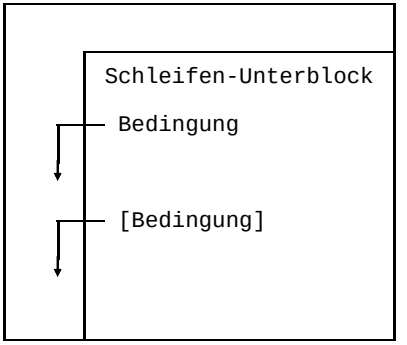
```
Name      | Operation | Operanden
----------|-----------|-----------------
LOOP      | @WHIL     | EQ
          | CLI       | END,C'N'
          | @DO       |
          | .         |
          | .         |
          | @BEND     |
```

Der Schleifenunterblock wird ausgeführt, solange END gleich "N" ist.

9.2.3.2 Schleife mit freier Endebedingung

Bei der Schleife mit freier Endebedingung können eine oder mehrere Endebedingungen an beliebigen Stellen des Schleifen-Unterblocks angegeben werden.

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@CYCL . . .	
[name]	@WHEN	cond_sym [Anzeige-setzender Assemblerbefehl]
[name]	@BREA . . .	
[name]	@BEND	

cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol; gibt an, worauf die Bedingung abgefragt werden soll.

Beschreibung

Der Schleifen-Unterblock wird von @CYCL und @BEND begrenzt. Er wird so lange wiederholt, bis eine der Endebedingungen zutrifft. Dann wird die Schleife an der angegebenen Stelle mit @BREA verlassen.

Die gesetzten Bedingungen (siehe 9.2.4 und 9.2.5) werden jeweils bei ihrem Auftreten geprüft.

@WHEN-@BREA kann innerhalb des Schleifen-Unterblocks mehrfach gesetzt sein.

Innerhalb des Schleifen-Unterblocks können weitere Strukturblöcke geschachtelt sein. Diese dürfen nur dann mit @WHEN-@BREA verlassen werden, wenn sie ebenfalls @CYCL-Schleifen sind. Bei so geschachtelten @CYCL-Schleifen beendet eine @WHEN-@BREA-Bedingung nur jeweils die Schleife, in der sie definiert ist, nicht den gesamten Strukturblock.

Beispiel

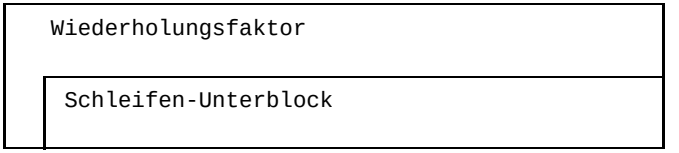
Register 3 enthält die Adresse einer Zeichenkette. In dieser soll die Position des Trennzeichens '*' gesucht werden.

```
Name    | Operation | Operanden
--------|-----------|-----------
        | @CYCL     |
        | @WHEN     | EQ
        | CLI       | 0(R3),C'*'
        | @BREA     |
        | AH        | R3,ONE
        | @BEND     |
        | .         |
        | .         |
ONE     | DC        | H'1'
```

9.2.3.3 Zählschleife

Bei einer Zählschleife wird der Unterblock so oft wiederholt, wie der Wiederholungsfaktor angibt.

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@CYCL Schleifen-Unterblock	(reg)
[name]	@BEND	

reg Register, dessen Inhalt die Anzahl der Wiederholungen angibt

Beschreibung

reg wird nach jedem Durchlauf um 1 vermindert und auf 0 abgefragt. Ist 0 erreicht, wird der Strukturblock verlassen.

Ist der Inhalt von reg schon am Anfang kleiner als 1,
- wird die Schleife sofort beendet, falls im Prozedurkopf CHECK=ON gesetzt ist (siehe 10.2, @ENTR),
- entsteht eine Endlosschleife, falls CHECK=OFF gesetzt ist und die Schleife keine Endebedingung hat.

reg darf im Schleifen-Unterblock nicht verändert werden.

Beispiel

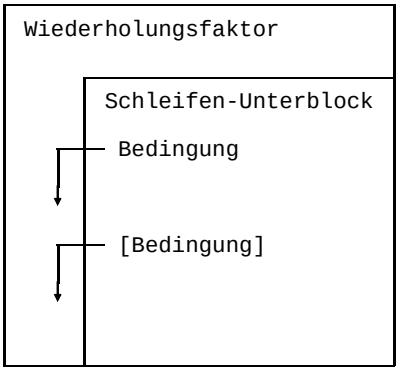
Die Tabelle 2 (TAB2) soll in die Tabelle 1 (TAB1) kopiert werden. Die Anzahl der Elemente ist im Feld N angegeben, die Länge jedes Elements in L.

```
Name     | Operation | Operanden
---------+-----------+-----------------
         | L         | R5,N
         | LA        | R6,TAB1-L
         | LA        | R7,TAB2-L
LOOP     | @CYCL     | (R5)
         | LA        | R6,L(0,R6)
         | LA        | R7,L(0,R7)
         | MVC       | 0(L,R6),0(R7)
         | @BEND     |
```

9.2.3.4 Zählschleife mit freier Endebedingung

Die Zählschleife mit freier Endebedingung vereinigt in sich die Eigenschaften der Zählschleife mit denen der Schleife mit freier Endebedingung. Es gibt nebeneinander @WHEN-@BREA-Endebedingungen und einen Wiederholungsfaktor. Der Wiederholungsfaktor bildet eine obere Schranke für die Zahl der Durchläufe des Schleifenunterblocks.

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@CYCL . . .	(reg)
[name]	@WHEN [Anzeige-setzender Assemblerbefehl]	cond_sym
[name]	@BREA . . .	
[name]	@BEND	

reg Register, dessen Inhalt die Anzahl der Wiederholungen angibt

cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol; gibt an, worauf die Bedingung abgefragt werden soll.

Beschreibung

Der Schleifen-Unterblock wird von @CYCL und @BEND begrenzt. Er wird so oft wiederholt, wie der Wiederholungsfaktor angibt, bzw. bis eine der Endebedingungen zutrifft. Dann wird die Schleife an der angegebenen Stelle mit @BREA verlassen.

Die gesetzten Bedingungen werden jeweils bei ihrem Auftreten geprüft.

@WHEN-@BREA kann innerhalb des Schleifen-Unterblocks mehrfach gesetzt sein.

Innerhalb des Schleifen-Unterblocks können weitere Strukturblöcke geschachtelt sein. Diese dürfen nur dann mit @WHEN-@BREA verlassen werden, wenn sie ebenfalls @CYCL-Schleifen sind. Bei so geschachtelten @CYCL-Schleifen beendet eine @WHEN-@BREA-Bedingung nur jeweils die Schleife, in der sie definiert ist, nicht den gesamten Strukturblock.

reg wird nach jedem Durchlauf um 1 vermindert und auf 0 abgefragt. Ist 0 erreicht, wird der Strukturblock verlassen.

reg darf im Schleifen-Unterblock nicht verändert werden.

Beispiel

Die Schleife im Beispiel wird zehnmal durchlaufen, wenn nicht vorher die Abbruchbedingung eintritt.

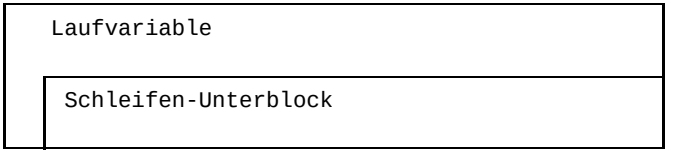
```
Name     | Operation | Operanden
---------+-----------+-----------------
         | L         | R7,NR              (01)
LOOP     | @CYCL     | (R7)               (02)
         | .         |
         | .         |
         | @WHEN     | EQ                 ⎫
         | CLC       | TEST,END           ⎬ (03)
         | @BREA     |                    ⎭
         | .         |
         | .         |
         | @BEND     |
         | .         |
         | .         |
NR       | DC        | F'10'
TEST     | .         | .
END      | .         | .
```

(01) Register 7 wird mit dem Wiederholungsfaktor geladen
(02) Schleifenanfang
(03) Abbruchbedingung und Schleifenausgang

9.2.3.5 Iterative Schleife

In einer iterativen Schleife bestimmen Anfangs- und Endwert der Laufvariablen die Zahl der Wiederholungen des Schleifen-Unterblocks.

Darstellung nach Nassi-Shneiderman



Format des Strukturblocks

Name	Operation	Operanden
[name]	@THRU	(reg1),{(reg2) / name / literal}[,{(reg3) / name / literal}]
[name]	@DO Schleifen-Unterblock	
[name]	@BEND	

(reg1) Register; gibt den Anfangswert für die Laufvariable an

(reg2), bzw. name oder literal im zweiten Operanden
Register, Name einer Konstanten oder Literal, das den Endwert für die Laufvariable angibt. Die Konstante, bzw. das Literal muß dabei auf Halbwort ausgerichtet und ein Halbwort lang sein.

(reg3), bzw. name oder literal im dritten Operanden
Register, Name einer Konstanten oder Literal, das die Schrittweite angibt. Die Konstante, bzw. das Literal muß dabei auf Halbwort ausgerichtet und ein Halbwort lang sein.

Beschreibung

Der erste Operand von @THRU (reg1) muß vor dem Aufruf mit dem Anfangswert der Laufvariablen geladen werden.

Vor jeder Ausführung des Schleifen-Unterblocks wird abgefragt, ob die Summe aus Laufvariabler und Schrittweite den Endwert übersteigt. Ist dies der Fall, wird der Strukturblock verlassen.

Nach jeder Ausführung des Schleifen-Unterblocks wird die Schrittweite zum Anfangswert, bzw. zum aktuellen Wert der Laufvariablen addiert. Der aktuelle Wert der Laufvariablen liegt jeweils in reg1 vor.

Der Schleifen-Unterblock wird so oft wiederholt, wie der ganzzahlige Teil des folgenden Ausdrucks angibt:

```
((Endwert-Anfangswert)/Schrittweite)+1
```

Wird keine Schrittweite angegeben, darf Register 0 nicht für den Anfangswert verwendet werden.

Ein @THRU-Makro mit Anfangswert ≤ Endwert und Schrittweite ≤ 0 führt zu einer Endlosschleife.

Beispiel

```
Name    | Operation | Operanden
--------+-----------+-----------------
        | .         |
        | .         |
        | LH        | R6,BEGL
LOOP    | @THRU     | (R6),ENDL,PLUS
        | @DO       |
        | .         |
        | .         |
        | @BEND     |
        | .         |
        | .         |
BEGL    | DC        | H'1'
ENDL    | DC        | H'10'
PLUS    | DC        | H'2'
```

In diesem Beispiel ist der Anfangswert der Laufvariablen 1, der Endwert 10 und die Schrittweite 2. Daher wird der Schleifenunterblock 5 mal durchlaufen.

9.2.4 Einfache Bedingungen

Format

Name	Operation	Operanden
		cond_sym [Anzeige-setzender Assemblerbefehl]

cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe unten); gibt an, worauf die Bedingung abgefragt werden soll

Anzeige-setzender Assemblerbefehl
Befehl, der unmittelbar vor der Abfrage der Anzeige ausgeführt wird. Der Befehl, der die abzufragende Anzeige setzt, muß nicht unmittelbar nach dem Bedingungssymbol folgen. Steht er außerhalb der Bedingung, darf die Anzeige zwischen dem Setzen und der Abfage nicht durch einen weiteren Befehl geändert werden. Die Abfrage der Anzeige erfolgt durch @THEN, @DO und @BREA.

9.2.4.1 Vordefinierte Bedingungssymbole

Es gibt bei der strukturierten Programmierung eine Reihe vordefinierter Bedingungssymbole. Sie werden entsprechend der folgenden Tabelle den am häufigsten vorkommenden Vergleichsergebnissen zugeordnet.

Ergebnisse von Vergleichsbefehlen

=	EQ	equal
<	LT	less than
>	GT	greater than
≠	NE	not equal
≥	GE	greater or equal
≤	LE	less or equal

Eigenschaften von Ergebnissen arithmetischer und logischer Operationen

=0	ZE	zero
<0	LZ	less than zero
>0	GZ	greater than zero
≠0	NZ	not zero
Überlauf	ON	

Ergebnisse des TM-Befehls (Testen mit Maske)

nur binäre Einsen	ON	ones
nur binäre Nullen	ZE	zeroes
gemischt Einsen und Nullen	MI	mixed
nur Nullen oder nur Einsen	ZO	ZE oder ON
mindestens eine binäre Null	ZM	ZE oder MI
mindestens eine binäre Eins	OM	ON oder MI

Tabelle 9-1 Zuordnung der vordefinierten Bedingungssymbole zu den am häufigsten vorkommenden Befehlsergebnissen

Beispiele

```
Name        | Operation | Operanden
------------+-----------+-----------
            | @...      | LZ
            | LTR       | R1,R1
```

Die Bedingung ist wahr, wenn der Inhalt des Registers R1 negativ (less than zero) ist. Der Befehl LTR R1,R1 bewirkt nichts anderes als das Setzen der Anzeige.

```
            | @...      | GZ
            | S         | R1,ALPHA
```

Die Bedingung ist wahr, wenn das Ergebnis der Subtraktion Register R1 minus ALPHA größer als Null (greater than zero) ist, sonst falsch. Außer dem Setzen der Anzeige wird auch der Registerinhalt verändert.

```
            | @...      | ON
            | TM        | 0(RPEGEL),MASKE
```

Die Bedingung ist wahr, wenn alle durch die Maske MASKE ausgewählten Bits des Bytes mit der im Register RPEGEL stehenden Adresse binäre Einsen (ones) sind.

```
            | @...      | EQ
            | CLC       | KANDIDAT,0(RLISTE)
```

Die Bedingung ist wahr, wenn der Inhalt von KANDIDAT gleich (equal) dem gleichlangen Wert ist, der über das Register RLISTE adressiert ist.

9.2.4.2 Benutzereigene Bedingungssymbole

Zusätzlich zu den vordefinierten Bedingungssymbolen kann der Benutzer den möglichen Bedingungsmasken eigene Bedingungssymbole zuweisen. Diese müssen jeweils mit einem '@' anfangen.

Format der Zuweisung

Name	Operation	Operanden
@name	EQU	Maske

Der Aufbau der Maske im Operandeneintrag der EQU-Anweisung ist aus der folgenden Tabelle zu ersehen.

(Die in einer Zeile stehenden Bedingungssymbole sind gleichwertig).

Anzeige				Maske für die bedingte Verzweigung		entspricht dem vordef. Bedingungssymbol
0	1	2	3	binär	numerisch	
gesetzt				1000	8	EQ ZE
	gesetzt			0100	4	LT LZ MI
		gesetzt		0010	2	GT GZ
			gesetzt	0001	1	ON
nicht gesetzt				0111	7	NE NZ OM
	nicht gesetzt			1011	11	GE ZO
		nicht gesetzt		1101	13	LE
			nicht gesetzt	1110	14	ZM

Tabelle 9-2 Masken für die Zuweisung benutzereigener Bedingungssymbole

9.2.5 Zusammengesetzte Bedingungen

Eine zusammengesetzte Bedingung besteht aus zwei oder mehr einfachen Bedingungen, die mit Hilfe der Makros @TOR, @AND und @OR verknüpft werden müssen.

Format

Name	Operation	Operanden
[name]	@IF @WHEN @WHIL Anzeige-setzender Assemblerbefehl	cond_sym
[name]	@TOR @AND @OR Anzeige-setzender Assemblerbefehl	cond_sym

cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe 9.2.4)

Anzeige-setzender Assemblerbefehl
Befehl, der unmittelbar vor der Abfrage der Anzeige ausgeführt wird.

In einer zusammengesetzten Bedingung muß jede einfache Bedingung den Anzeige-setzenden Assemblerbefehl enthalten.

Unter Umständen werden nicht alle Anzeige-setzenden Befehle ausgeführt. Wenn z.B. in einer Und-Verknüpfung dreier einfacher Bedingungen die erste Bedingung als nicht erfüllt erkannt wurde, bleiben die beiden restlichen Bedingungen einschließlich der in ihnen enthaltenen Befehle unbeachtet. Es empfiehlt sich daher, in zusammengesetzten Bedingungen nur solche Befehle zu verwenden, deren einzige Wirkung im Setzen der Anzeige besteht.

In einer zusammengesetzten Bedingung realisiert
- @TOR das logische "Oder mit Priorität",
- @AND das logische "Und" und
- @OR das logische "Oder".

Die Bindungspriorität der Makros, die logische Operatoren darstellen, entspricht der aufgeführten Reihenfolge. @TOR realisiert dabei, ebenso wie @OR, die Oder-Verknüpfung, hat aber höhere Bindungspriorität. D.h., durch die Verwendung von @TOR statt @OR wird die Reihenfolge der Bearbeitung gegenüber @AND geändert.

Beispiele

```
Name        | Operation | Operanden
------------+-----------+-----------------------
            | @CYCL     |
            | @WHEN     | EQ                    Bedingung 1
            | CR        | R1,R2
            | @AND      | EQ                    Bedingung 2
            | C         | R3,N
            | @TOR      | EQ                    Bedingung 3
            | C         | R4,M
            | @BREA     |
            | .         |
            | .         |
```

Die Schleife wird verlassen, wenn einer der beiden Fälle gegeben ist:
- Bedingung 1 und Bedingung 2 treffen zu
- Bedingung 1 und Bedingung 3 treffen zu.

```
Name        | Operation | Operanden
------------+-----------+-----------------------
            | @CYCL     |
            | @WHEN     | EQ                    Bedingung 1
            | CR        | R1,R2
            | @AND      | EQ                    Bedingung 2
            | C         | R3,N
            | @OR       | EQ                    Bedingung 3
            | C         | R4,M
            | @BREA     |
            | .         |
            | .         |
```

Das zweite Beispiel ist gleich dem ersten, bis auf die Verwendung von @OR statt @TOR in Bedingung 3.

Die Schleife wird hier verlassen, wenn einer der beiden Fälle gegeben ist:
- Bedingung 1 und Bedingung 2 treffen zu
- nur Bedingung 3 trifft zu.

9.3 Prozedurkonzept

Im Prozedurkonzept wird der Kontrollfluß zwischen Hauptprogrammen und Unterprogrammen geregelt. Eine Prozedur im Sinne der Strukturierten Programmierung ist eine Programmeinheit mit einem Eingang und einem Ausgang, die unter einem Namen, ggf. mit aktuellen Parameterwerten, aufgerufen werden kann. Für einen Prozedurtext können - im Gegensatz zu den Strukturblöcken - das dynamische und das statische Ende unterschiedlich sein.

Eine Prozedur ist aus aneinandergereihten oder geschachtelten Strukturblöcken aufgebaut. Wird ein Strukturblock zu kompliziert oder zu umfangreich, kann man ihn auslagern und zu einer eigenen Prozedur erklären.
Eine Prozedur kann in einen eigenen Objektmodul übersetzt werden. Es können aber auch zwei oder mehr Prozeduren in einem Objektmodul vorliegen.
Ein Prozedur kann eine andere aufrufen. Beim Aufruf kann die rufende Prozedur Parameter an die aufgerufene übergeben. Bei der Rückkehr in die rufende Prozedur kann die aufgerufene einen Rückkehrwert abgeben. Die rufende Prozedur wird mit der Instruktion fortgesetzt, die dem Aufruf unmittelbar folgt.

Prozeduren können nicht statisch ineinandergeschachtelt werden.

Die strukturierte Programmierung verlangt, daß Programme aus reentrantfähigen Prozeduren aufgebaut sind.
Eine Prozedur wird "reentrantfähig" genannt, wenn sich zu gleicher Zeit mehrere Anwenderprogramme auf dem Lauf durch den (einmal vorhandenen) Code der Prozedur befinden können. Ist dies der Fall, dann belegt ein Programm den Prozessor, die übrigen befinden sich im Wartezustand; ihre Registerstände sind in einem Registersicherungsbereich zwischengespeichert.

Die Datenbereiche, die Teil des Codes einer reentrantfähigen Prozedur sind, dürfen also nur Konstanten enthalten; Datenbereiche für Variable müssen zur Laufzeit separat angefordert und verwaltet werden.

Bei der strukturierten Programmierung wird unterschieden zwischen
- Datenbereichen, die beim Laden des Programms bereitgestellt werden und während des ganzen Programmlaufs erhalten bleiben (Static-Bereiche),
- Bereichen, die während des Programmlaufs angefordert und auch wieder freigegeben werden (Controlled-Bereiche) und
- Bereichen, die während eines Prozedurdurchlaufs erhalten bleiben (Automatic-Bereiche).

Sollen die erzeugten Prozeduren reentrantfähig sein, dürfen statisch bereitgestellte Datenbereiche nur der Speicherung von Konstanten dienen.

Für jeden Datenbereich, auf den in einer Prozedur zugegriffen wird, muß eine Definition in der Prozedur enthalten sein, die über die Speicherklasse und die Zugriffsart Auskunft gibt.

9.3.1 Prozedurerklärung und Prozedurende

Eine Prozedur wird statisch abgegrenzt durch die Makros @ENTR (Prozeduranfang) und @END (statisches Prozedurende). Beim Programmlauf wird eine Prozedur mit @PASS aufgerufen und mit @EXIT (dynamisches Prozedurende) beendet. Die folgende Abbildung zeigt die dynamische Verknüpfung von mehreren Prozeduren durch den Aufruf mit @PASS und die Rückkehr in die rufende Prozedur mit @EXIT.

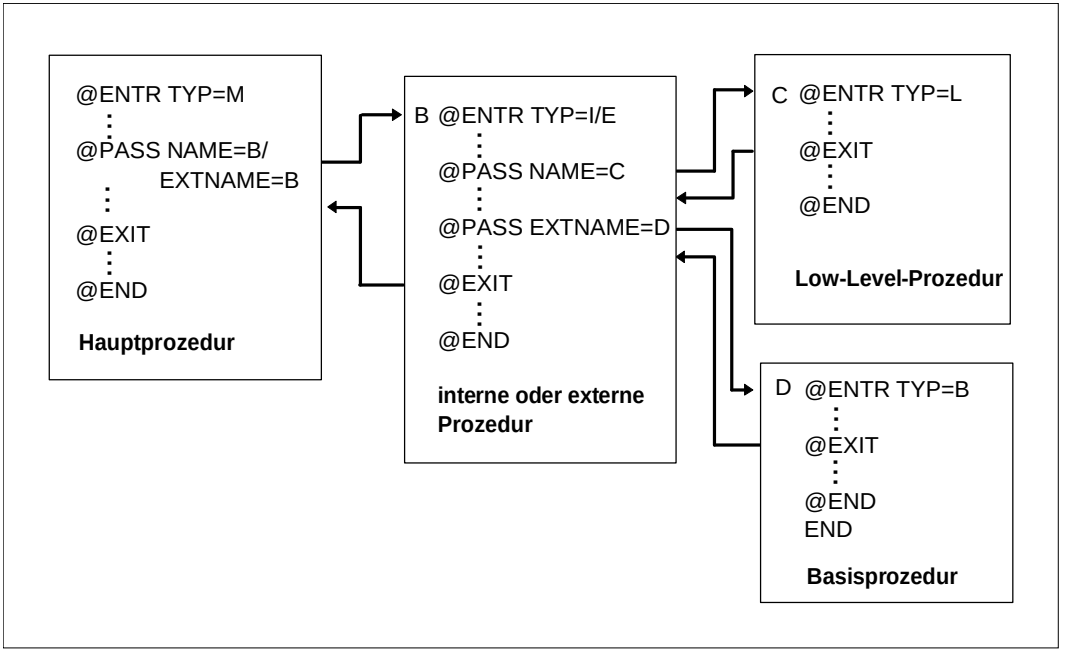


Bild 9-1 Dynamische Prozedurverknüpfung

Prozedurerklärung

Der @ENTR-Makro stellt den Kopf für alle Typen von Prozeduren dar. Hier muß neben dem Namen der Prozedur der Prozedurtyp angegeben werden.

Neben der Hauptprozedur, die vom Betriebssystem aus durch /START-PROGRAM oder /LOAD-PROGRAM aufgerufen wird (Typ M), werden externe und interne Prozeduren unterschieden. Eine Prozedur heißt "extern", wenn sie von einem beliebigen Modul aus, "intern", wenn sie nur innerhalb eines Moduls aufgerufen werden kann.

Externe Prozeduren können zu den Typen E, B, D, interne Prozeduren zu den Typen I, L, D gehören.

Der Aufruf von @ENTR gibt außerdem Auskunft darüber, ob die Prozedur Parameter von einer anderen übernehmen und/oder selbst Parameter an eine andere übergibt. Gekennzeichnet ist zusätzlich die Anzahl der Parameter und die Übergabeform, sowie die Datenbereiche, die hierfür bereit gestellt werden müssen (siehe 9.5 und 10.2, @ENTR).

Prozedurende

Hier muß unterschieden werden zwischen dem statischen und dem dynamischen Prozedurende.

Das statische Prozedurende wird realisiert durch den @END-Makro. Durch diesen Makro werden alle Basis- und Adressierungs-Register freigegeben, die für diese Prozedur durch den ASSEMBH zugewiesen wurden.

Der @EXIT-Makro kennzeichnet das dynamische Prozedurende. In einer Hauptprozedur wird durch @EXIT das Programm beendet.

In einer gerufenen Prozedur wird die Steuerung an die rufende Prozedur zurückgegeben. Diese wird mit der Instruktion fortgesetzt, die dem Prozeduraufruf folgt. Es ist möglich, an die rufende Prozedur einen Rückkehrwert zu übergeben.

Der Rückkehrwert wird in der gerufenen Prozedur berechnet. Im @EXIT-Makro wird entweder seine Adresse angegeben oder der Rückkehrwert direkt in Form eines selbstdefinierenden Wertes übergeben (siehe 10.2, @EXIT). In der rufenden Prozedur kann dieser Wert dann weiter verarbeitet oder ausgewertet werden.

9.3.2 Prozedurtypen

9.3.2.1 Prozeduren der Typen M, E und I

Prozeduren vom Typ M, E und I sind an die Speicherverwaltung durch das Laufzeitsystem angeschlossen. Bei ihrer Eröffnung werden automatisch die Register gesichert und bei ihrer Beendigung wieder zurückgeladen. Speicherbereiche für die Registersicherung und für zur Laufzeit angeforderte Daten werden dynamisch bereitgestellt.

In diesen Prozeduren können neben Datenbereichen der Klassen S und B (Static und Based) auch Datenbereiche der Klassen A und C (Automatic und Controlled) angefordert werden (siehe 9.4, Datenkonzept).

Für alle drei Prozedurtypen ist als Basisadreßregister das Register 10 zugewiesen. Dieses kann auch mit R@BASE angesprochen werden.

Prozeduren vom Typ M

Eine Prozedur vom Typ M ist die Hauptprozedur eines Programms. In ihr beginnt und endet die Programmausführung. Durch den @ENTR-Makro wird eine CSECT-Anweisung mit dem Namen des Programms generiert. Auch wenn Prozeduren aus mehreren Moduln zu einem Programm zusammengebunden werden, darf nur eine Prozedur vom Typ M vorhanden sein.

Eine Prozedur vom Typ M kann nicht von einer anderen aufgerufen werden. Sie kann also auch keine Parameter von einer anderen Prozedur übernehmen.

Die Übergabe von Parametern von der Hauptprozedur an weitere Prozeduren kann statisch oder dynamisch, über die STANDARD- oder die OPTIMAL-Schnittstelle erfolgen (siehe 9.5, Prozedurverknüpfung und Parameterübergabe).

Prozeduren vom Typ E und I

Eine Prozedur vom Typ E kann von einem beliebigen Modul aus aufgerufen werden (externe Prozedur).

Eine Prozedur vom Typ I kann nur innerhalb des Moduls aufgerufen werden, in dem sie liegt (interne Prozedur).

Prozeduren beider Typen können weitere interne oder externe Prozeduren aufrufen. Sie können sowohl Parameter an die gerufene Prozedur übergeben, als auch selbst Parameter von einer rufenden Prozedur übernehmen. Die Übergabe der Parameter ist sowohl statisch als auch dynamisch, über die STANDARD- oder die OPTIMAL-Schnittstelle möglich, die Übernahme von Parametern erfolgt nur statisch (siehe 9.5, Prozedurverknüpfung und Parameterübergabe).

Bei der Rückkehr in die rufende Prozedur werden die durch ASSEMBH zugewiesenen Register wieder mit den ursprünglichen Werten geladen. Über den Operanden RETURNS= im @ENTR-Makro kann diese Möglichkeit erweitert oder eingeschränkt werden (siehe @ENTR, Format 2).

Durch den @ENTR-Makro wird bei E-Prozeduren standardmäßig eine CSECT-Anweisung mit dem Prozedurnamen generiert. Parametergesteuert kann auch nur eine ENTRY-Anweisung statt der CSECT-Anweisung generiert werden.

In I-Prozeduren wird keine CSECT-Anweisung generiert. D.h., sie gehören jeweils zum vorausgehenden Programmabschnitt. Der Anwender muß darauf achten, daß eine I-Prozedur nicht im Bereich eines Pseudoabschnitt (DSECT-Anweisung) liegt.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-------------------
MAIN      | @ENTR     | TYP=M
          | @DATA     | ...,DSECT=DUMMY,...
          | .         |
          | .         |
          | @EXIT     |
          | @END      |
DUMMY     | DSECT     |
          | .         |
          | .         |
LDUMMY    | EQU       | *-DUMMY
MAIN      | CSECT     |
INTERN    | @ENTR     | TYP=I
          | .         |
          | .         |
```

9.3.2.2 Prozeduren der Typen B, L und D

Prozeduren vom Typ B, L und D sind nicht an die Speicherverwaltung durch das Laufzeitsystem angeschlossen. Es werden keine Speicherbereiche für die Registersicherung oder eine dynamische Parameterübergabe bereitgestellt. Die Register werden bei der Eröffnung dieser Prozeduren nicht gesichert und bei ihrer Beendigung auch nicht zurückgeladen.

Die Typen B, L und D sind für Prozeduren vorgesehen, die in der Aufrufhierarchie auf einer niedrigeren Ebene liegen und dementsprechend Basisfunktionen ausführen.

In diesen Prozeduren muß der Anwender selbst für die Speicherverwaltung sorgen. Es dürfen keine Speicheranforderungen (Datenbereiche der Klassen A und C, siehe 9.4) gestellt werden.

Prozeduren der Typen B und L

Eine Prozedur vom Typ B (Basisprozedur) kann von einem beliebigen Modul aus aufgerufen werden.

Eine Prozedur vom Typ L (Low-level-Prozedur) kann nur innerhalb des Moduls aufgerufen werden, in dem sie liegt.

Für beide Typen ist standardmäßig Register 15 als Basisadreßregister zugewiesen. Im BASE-Operanden des @ENTR-Makro kann der Anwender ein anderes Register als Basisadreßregister zuweisen. Dieses Register muß am Anfang der Prozedur explizit mit dem richtigen Wert geladen werden (z.B mit dem Assemblerbefehl "LR reg,R15").

Das Laden des Basisadreßregisters nach jedem Aufruf einer weiteren Unterprozedur wird über den Operanden LOADSB= im @ENTR-Makro gesteuert. Die Angabe von LOADSB=YES ist nur notwendig, wenn die Prozedur ein Ersatzbasisregister hat und eine andere Prozedur aufruft.

Prozeduren vom Typ B und L können ihrerseits weitere externe oder interne Prozeduren aufrufen. Eine Parameterübergabe an die gerufene Prozedur ist nur statisch möglich.

In Prozeduren vom Typ B wird durch den @ENTR-Makro standardmäßig eine CSECT-Anweisung mit dem Prozedurnamen generiert. Parametergesteuert kann auch nur eine ENTRY-Anweisung statt der CSECT-Anweisung generiert werden.

Bei Prozeduren vom Typ L wird keine CSECT-Anweisung generiert. D.h., sie gehören jeweils zum vorausgehenden Programmabschnitt. Der Anwender muß darauf achten, daß eine L-Prozedur nicht im Bereich eines Pseudoabschnitts (DSECT-Anweisung) liegt.

Prozeduren vom Typ D

Prozeduren vom Typ D (Pseudoprozeduren) sind für einfache Abläufe vorgesehen, die ohne viel Aufwand erledigt werden können. Bei diesem Typ muß auch die Basisregisterversorgung vom Anwender explizit durchgeführt werden. D-Prozeduren können sowohl extern als auch intern sein.

In einer Prozedur vom Typ D ist **nicht** erlaubt:
- der Aufruf von weiteren Unterprozeduren mit @PASS,
- die Rückkehr in die rufende Prozedur mit @EXIT,
- Speicheranforderung und Speicherfreigabe.

Ist die erste Prozedur eines Moduls vom Typ D, darf kein Prozedurname angegeben werden. Die Prozedur erhält den Namen der START-, bzw. CSECT-Anweisung.

9.4 Datenkonzept

Das Datenkonzept von ASSEMBH verhilft dem Anwender zur rationellen Speichernutzung und nimmt ihm die Speicherverwaltung ab, die für die Erzeugung reentrant-fähiger Prozeduren notwendig ist. Das Datenkonzept regelt den Gültigkeitsbereich aller Daten und sorgt für Übersichtlichkeit. Es verlangt, daß jede Prozedur über alle verwendeten Daten Auskunft gibt. In einer Prozedur muß angegeben werden, welche Daten sie benutzt, wo sich die Daten befinden und wie der Zugriff organisiert ist.

Für reentrant-fähige Prozeduren gilt:
- Datenbereiche, die Teil einer Prozedur sind, dürfen nur Konstanten enthalten.
- Datenbereiche für Variable müssen zur Laufzeit angefordert werden.

Dementsprechend wird unterschieden zwischen Datenbereichen, die beim Laden des Programms bereitgestellt werden und während des gesamten Programmlaufs zur Verfügung stehen, und Bereichen, die während des Programmlaufs angefordert und wieder freigegeben werden.

Bei den dynamisch angeforderten Bereichen wird unterschieden zwischen solchen, die bei Verlassen der Prozedur, in der die Anforderung erfolgte, automatisch wieder freigegeben werden, und solchen, die benutzergesteuert (controlled) durch eine explizite Instruktion wieder freigegeben werden.

Gemäß dieser Unterscheidung lassen sich die Speicherbereiche eines Programms in die folgenden vier Klassen einteilen:
- Static,
- Automatic,
- Controlled und
- Based.

Die Zuordnung einer Klasse zu einem Bereich legt der Anwender im @DATA-Makro fest. Mit dem @DATA-Makro wird für Static- und Based-Bereiche der Zugriff realisiert, für Automatic- und Controlled-Bereiche Speicherplatz angefordert und der Zugriff realisiert.

Bereiche der Klasse Static werden durch "@DATA CLASS=S" (siehe 10.2, Format 2 von @DATA) zugeordnet. Sie erhalten während des Ladens des Programms Speicherplatz zugewiesen, der während des gesamten Programmlaufs bestehen bleibt. Der @DATA-Makro lädt dazu ein Basisadreßregister.

Für Bereiche der Klasse Automatic wird entweder mit "@DATA CLASS=A" (siehe 10.2, Format 1 von @DATA) Speicher bereitgestellt oder über die Anforderung eines LOCAL-Bereichs (siehe 9.4.2). Der Speicherbereich wird innerhalb des STACK-Bereiches der Prozedur bereitgestellt. Er wird automatisch wieder freigegeben, wenn die Prozedur, in der er angefordert wurde, durch @EXIT verlassen wird.

Für Bereiche der Klasse Controlled wird durch "@DATA CLASS=C" (siehe 10.2, Format 1 von @DATA) ein Bereich im HEAP-Speicher bereitgestellt. Er wird erst dann freigegeben, wenn der Benutzer dies ausdrücklich durch den @FREE-Makro fordert. Dies kann auch in einer anderen Prozedur erfolgen.

Soll ein Bereich, der in einer Prozedur einer der Klassen Static, Automatic oder Controlled zugeordnet wurde, auch in einer untergeordneten oder zeitlich später zu durchlaufenden Prozedur angesprochen werden, so muß er dort mit der Angabe eines Registers, welches die Adresse des Bereiches enthält, durch "@DATA CLASS=B" (siehe 10.2, Format 3 von @DATA) bekannt gemacht werden. Dadurch wird ihm für diese Prozedur das Klassenmerkmal "Based" zugeordnet.

Register 0, 1, 13, 14, 15 und das Prozedurbasisregister dürfen als Basisadreßregister für einen durch @DATA definierten Datenbereich nicht benützt werden.

9.4.1 Datenbereiche der Klasse Static

Datenbereiche der Klasse Static sind für konstante Daten vorgesehen. Der Zugriff auf Static-Bereiche wird mit "@DATA CLASS=S" realisiert. Diese Bereiche werden statisch im Programm definiert, so daß bereits beim Laden des Programms Speicherplatz reserviert wird, der bis zum Programmende bestehen bleibt. Die Daten in einem Static-Bereich stehen während des ganzen Programmlaufs zur Verfügung.

Der Speicherplatz für Static-Bereiche muß entweder
- intern nach dem statischen Ende der Prozedur (nach @END und vor @ENTR, bzw. der END-Anweisung) oder
- extern in einem eigenen Datenmodul bereitgestellt werden.

Als Basisadreßregister ist das Register zugewiesen, daß im BASE-Operanden von @DATA angegeben ist.

Interne Static-Bereiche

Der INIT-Operand von @DATA gibt die symbolische Adresse des Datenbereichs an.

Beispiel

```
Name     | Operation | Operanden
---------|-----------|-----------------------------------
EX1      | @ENTR     | TYP=E
         | .         |
         | .         |
         | @DATA     | CLASS=S,BASE=R5,INIT=CONST
         | .         |
         | .         |
         | @EXIT     |
         | @END      |
CONST    | DS        | 0CL180
C1       | DC        | C'ABC'
         | .         |
         | .         |
```

Für den Static-Bereich in der Prozedur EX1 wird Register 5 als Basisadreßregister zugewiesen. Dieser Bereich beginnt an der symbolischen Adresse CONST.

Externe Static-Bereiche

Der EXTINIT-Operand von @DATA gibt die symbolische Adresse eines Datenbereichs an, der in einem eigenen Datenmodul liegt. In der zugreifenden Prozedur muß die Struktur dieses Bereichs mit Hilfe eines Pseudoabschnitts beschrieben werden. Den Namen dieses Pseudoabschnitts gibt der Operand DSECT= an.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------------------------
* Zugreifende Prozedur
EX2         | @ENTR     | TYP=I
            | .         |
            | .         |
            | @DATA     | CLASS=S,BASE=R5,EXTINIT=EXDATEN,DSECT=EX2DUMMY
            | .         |
            | .         |
            | @EXIT     |
            | @END      |
EX2DUMMY    | DSECT     |
E1          | DS        | CL5
            | .         |
            | .         |
EEND        | DS        | CL3

* Datenmodul
EXDATEN     | CSECT     |
            | @ENTR     | TYP=D
D           | DS        | 0CL180
D1          | DC        | C'HALLO'
            | .         |
            | .         |
DEND        | DC        | C'END'
            | @END      |
```

Als Basisadreßregister für den Static-Bereich der Prozedur EX2 wird Register 5 zugewiesen. Der Datenbereich beginnt an der Adresse EXDATEN. Der Pseudoabschnitt EX2DUMMY beschreibt in der Prozedur EX2 die Struktur des angesprochenen Datenbereichs.

9.4.2 Datenbereiche der Klasse Automatic

Datenbereiche der Klasse Automatic enthalten veränderbare Daten. Der Speicherplatz wird von einer Prozedur zur Laufzeit angefordert und im Prozedur-STACK zur Verfügung gestellt. Dieser Prozedur-STACK wird beim Ansprung jeder Prozedur angelegt und beim Verlassen der Prozedur mit @EXIT automatisch wieder freigegeben (prozedurgebundene Lebensdauer).

Wird die Prozedur rekursiv aufgerufen oder von verschiedenen Anwendern "gleichzeitig" durchlaufen (reentrant), so bestehen mehrere Datenbereiche nebeneinander.

Datenbereiche der Klasse Automatic können nur in Prozeduren vom Typ M, E und I angefordert werden.

Es gibt zwei Möglichkeiten für das Anfordern eines Automatic-Bereichs:
- Der Datenbereich wird über das Basisregister 13 zusammen mit dem Registersicherstellungsbereich der Prozedur adressiert (Anfordern eines LOCAL-Bereichs).
- Der Datenbereich wird über ein eigenes Register, das der Benutzer explizit angibt, adressiert (Anfordern eines CLASS-A-Bereichs).

Anfordern eines LOCAL-Bereichs

Beim Eröffnen einer Prozedur wird automatisch der Prozedur-STACK bereitgestellt, der den Registersicherstellungsbereich (SAVAREA) und den LOCAL-Bereich enthält. Das Basisadreßregister für den Prozedur-STACK und damit auch für die SAVAREA und für den LOCAL-Bereich ist Register 13. Der Zugriff auf den LOCAL-Bereich ist nach Durchlaufen des Prozedurkopfes möglich.

Der Bereich wird angefordert über den LOCAL-Operanden von @ENTR (siehe 10.2, Format 1 und 2 von @ENTR). Eine Initialisierung des neu angeforderten Bereichs ist nicht möglich. Auf Daten in diesem Speicherbereich kann man nicht von einer anderen Prozedur aus zugreifen.

Die Struktur dieses Speicherbereichs wird von einem Pseudoabschnitt beschrieben, der unmittelbar nach dem statischen Ende der betreffenden Prozedur definiert sein muß. Der Pseudoabschnitt muß mit dem @PAR-Makro definiert werden (siehe 10.2, Format 3 von @PAR). Der Name des Pseudoabschnitts muß mit der Angabe im LOCAL-Operanden von @ENTR übereinstimmen.

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-------------------
EX1         | @ENTR     | TYP=I,LOCAL=PRIVATE
            | .         |
            | .         |
            | @EXIT     |
            | @END      |
PRIVATE     | @PAR      | D=YES              (01)
P1          | DS        | CL5
            | .         |
            | .         |
PRIVATE     | @PAR      | LEND=YES           (02)
```

(01) Beginn des Pseudoabschnitts
(02) Ende des Pseudoabschnitts

Anfordern mit @DATA CLASS=A

Für einen Datenbereich, der mit "@DATA CLASS=A" angefordert wurde, ist als Basisadreßregister das Register zugewiesen, das im BASE-Operanden von @DATA angegeben ist.

Der Speicherplatz wird angefordert entsprechend der Länge, die entweder im Operanden LENGTH= direkt angegeben wird oder die anhand eines zugeordneten Pseudoabschnitts errechnet wird. Bei Angabe von LENGTH= entfällt die Möglichkeit der symbolischen Adressierung.

Der zugeordnete Pseudoabschnitt beschreibt die Struktur des angeforderten Speicherbereichs. Seine Definition muß mit einer DSECT-Anweisung beginnen. Der Pseudoabschnitt muß abgeschlossen werden mit der folgenden Assembleranweisung:

```
Ldsect_name    EQU  *-dsect_name
```

Beispiel

```
Name        | Operation | Operanden
------------|-----------|-----------------------------
MAIN        | @ENTR     | TYP=M
            | @DATA     | CLASS=A,BASE=R7,DSECT=DUMMY
            | .         |
            | .         |
            | @EXIT     |
            | @END      |
DUMMY       | DSECT     |
D           | DS        | 0CL20
D1          | DS        | CL10
D2          | DS        | CL10
LDUMMY      | EQU       | *-DUMMY
```

Automatic-Bereiche, die mit @DATA CLASS=A angefordert wurden, können initialisiert werden. D.h., bereits definierte Daten werden in den angeforderten Speicherbereich kopiert. Diese Daten können im gleichen Modul liegen, wie die betreffende Prozedur, dann werden sie mit dem Operanden INIT= angesprochen. Liegen die zu kopierenden Daten in einem anderen Modul, ist der Name des Datenbereichs mit EXTINIT= anzugeben.

Beispiel

```
Name     | Operation | Operanden
---------+-----------+-----------------------------------------
MAIN     | @ENTR     | TYP=M
         | @DATA     | CLASS=A,BASE=R7,LENGTH=(R8),INIT=IDAT
         | .         |
         | .         |
         | @EXIT     |
IDAT     | DC        | C'ABC'
         | DC        | C'DEF'
         | .         |
         | .         |
         | @END      |
```

9.4.3 Datenbereiche der Klasse Controlled

Datenbereiche der Klasse Controlled enthalten veränderbare Daten. Der Speicherplatz wird von einer Prozedur zur Laufzeit angefordert und im HEAP-Speicher zur Verfügung gestellt. Er liegt außerhalb des Moduls und kann nur explizit mit @FREE vom Benutzer freigegeben werden (benutzergesteuerte Lebensdauer).

Wird die Prozedur rekursiv aufgerufen oder von verschiedenen Anwendern "gleichzeitig" durchlaufen (reentrant), so bestehen für jeden Durchlauf des @DATA-Makro mehrere Datenbereiche nebeneinander.

Datenbereiche der Klasse Controlled können nur in Prozeduren vom Typ M, E und I angefordert werden.

Datenbereiche der Klasse Controlled werden angefordert mit "@DATA CLASS=C" und adressiert über das Basisadreßregister, das im BASE-Operanden explizit angegeben ist.

Der Speicherplatz wird angefordert entsprechend der Länge, die entweder im Operanden LENGTH= direkt angegeben wird oder die anhand eines zugeordneten Pseudoabschnitts errechnet wird. Bei Angabe von LENGTH= entfällt die Möglichkeit der symbolischen Adressierung.

Der zugeordnete Pseudoabschnitt beschreibt die Struktur des angeforderten Speicherbereichs. Seine Definition muß mit einer DSECT-Anweisung beginnen. Der Pseudoabschnitt muß abgeschlossen werden mit der folgenden Assembleranweisung:

```
Ldsect_name    EQU  *-dsect_name
```

Controlled-Bereiche, die mit @DATA CLASS=C angefordert wurden, können initialisiert werden. D.h., bereits definierte Daten werden in den angeforderten Speicherbereich kopiert. Diese Daten können im gleichen Modul liegen, wie die betreffende Prozedur, dann werden sie mit dem Operanden INIT= angesprochen. Liegen die zu kopierenden Daten in einem anderen Modul, ist der Name des Datenbereichs mit EXTINIT= anzugeben.

Der Benutzer muß den angeforderten Speicherbereich mit @FREE freigeben (siehe 10.2). Das kann bereits vor Prozedurende geschehen, um Platz und Register zu sparen, oder in einer späteren Prozedur, falls z.B. die Daten noch weiter bearbeitet werden sollen.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------------------
MAIN      | @ENTR     | TYP=M
          | @DATA     | CLASS=C,BASE=R7,DSECT=DUMMY
          | .         |
          | .         |
          | @FREE     | BASE=R7
          | @EXIT     |
          | @END      |
DUMMY     | DSECT     |
D         | DS        | 0CL20
D1        | DS        | CL10
D2        | DS        | CL10
LDUMMY    | EQU       | *-DUMMY
```

9.4.4 Datenbereiche der Klasse Based

Die Klasse Based dient dem Zugriff auf einen Datenbereich, dessen Speicherplatz-Zuordnung in einer anderen, dynamisch übergeordneten oder zeitlich vorangegangenen Prozedur erfolgt ist (@DATA CLASS=S, CLASS=A oder CLASS=C).

Eine Unterprozedur greift auf einen bereits bestehenden Datenbereich zu mit @DATA CLASS=B (siehe 10.2, Format 3 von @DATA). Der BASE-Operand dieses Makroaufrufs gibt das Basisadreßregister an, das in der übergeordneten Prozedur für diesen Datenbereich zugewiesen wurde. Das Register muß in der übergeordneten Prozedur mit der Anfangsadresse des Bereichs versorgt werden. Dies geschieht entweder mit @DATA CLASS=A, C oder S oder z.B. bei der Parameterübergabe.

Um die symbolische Adressierung des angesprochenen Datenbereichs zu ermöglichen, ist im @DATA-Makro der Operand DSECT= anzugeben. Dieser enthält entweder:

- die symbolische Adresse des angesprochenen Datenbereichs oder
- den Namen eines Pseudoabschnitts, der die Struktur des Bereichs beschreiben soll. Der Pseudoabschnitt muß mit der folgenden Assembleranweisung abgeschlossen werden:

```
Ldsect_name     EQU   *-dsect_name
```

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-----------------------------------
EX1         | CSECT     |
DUMMY       | DSECT     |                                    ⎫
..          | DS        | ..                                 ⎪
            | .         |                                    ⎬ (01)
            | .         |                                    ⎪
LDUMMY      | EQU       | *-DUMMY                            ⎭

* Übergeordnete Prozedur
EX1         | @ENTR     | TYP=M
            | @DATA     | CLASS=S,BASE=R7,INIT=GLOB          (02)
            | .         |
            | .         |
            | @PASS     | NAME=EX2                           (03)
            | .         |
            | .         |
            | @EXIT     |
            | @END      |
GLOB        | DS        | 0C
..          | .         | ..
            | .         |

* Unterprozedur
EX2         | @ENTR     | TYP=I
            | @DATA     | CLASS=B,BASE=R7,DSECT=DUMMY        (04)
            | .         |
            | .         |
```

(01) Definition des Pseudoabschnitts

(02) Einrichten eines Static-Bereichs mit Register 7 als Basisadreßregister; der Bereich wird mit den Daten in GLOB initialisiert.

(03) Aufruf der Unterprozedur

(04) Zugriff auf den Datenbereich, der in EX1 eingerichtet wurde und der über R7 adressiert ist. Die Struktur des Speicherbereichs beschreibt für die Prozedur EX2 der Pseudoabschnitt DUMMY

9.5 Prozedurverknüpfung und Parameterübergabe

Die Verknüpfung von Prozeduren erfolgt mit Hilfe von @PASS (siehe 10.2). Dieser Makro ruft von einer dynamisch übergeordneten Prozedur aus eine Unterprozedur auf. Die gerufene Prozedur kehrt mit @EXIT in die rufende zurück.

Bei der Eröffnung von Prozeduren der Typen M, E und I wird automatisch der Prozedur-STACK bereitgestellt, der die Verkettungsinformation, die Register (SAVAREA) und evtl. den LOCAL-Bereich enthält. Der Prozedur-STACK wird über Register 13 adressiert. Die rufende Prozedur übergibt in Register 13 die Adresse des eigenen Prozedur-STACK, die gerufene Prozedur legt im Prozedur-STACK der rufenden die Register ab und lädt sie vor dem Rücksprung wieder.

Register 13 wird bei der Registersicherstellung nicht mitgesichert.

Beim Aufruf von Prozeduren der Typen B, L und D wird kein Prozedur-STACK bereitgestellt. Bei diesen Prozeduren muß der Anwender selbst für eine Registersicherung sorgen. Insbesondere muß bei Prozeduren vom Typ B oder L das Prozedurrückkehrregister 14 sichergestellt werden.

Parameterübergabe

Eine Prozedur kann beim Aufruf einer weiteren Parameter an die gerufene Prozedur übergeben. Ein Parameter ist immer ein Wort lang. Er kann eine Adresse oder einen Wert enthalten. An die gerufene Prozedur übergeben wird entweder der Parameter selbst oder seine Adresse. Für die Übergabe wird eine Parameterliste aufgebaut, die entsprechend für jeden Parameter dessen Adresse oder den Wert enthält.

Für die Parameterübergabe sind verschiedene Formen vorgesehen. Diese berücksichtigen

- ob die bei einem Aufruf zu übergebenden Parameterwerte sich schon beim Programmieren festlegen lassen (statische Übergabe) oder ob sie erst zur Laufzeit bekannt sind (dynamische Übergabe),
- ob bei dynamischer Übergabe die Parameteradressen in einer Liste zusammengestellt werden, deren Adresse in Register 1 übergeben wird (STANDARD-Schnittstelle) oder ob bis zu vier Parameteradressen oder -werte in den Registern 1 bis 4 übergeben werden (OPTIMAL-Schnittstelle).

Die zu einem @PASS aufgebaute Parameterliste besteht bei statischer Übergabe aus Konstanten, die zur Übersetzungszeit generiert werden und während der gesamten Lebensdauer des Programms erhalten bleiben. Bei dynamischer Übergabe werden die Parameterlisten wie veränderbare Daten behandelt. Ihr Speicherplatz wird im Prozedur-STACK der rufenden Prozedur bereitgestellt und beim Prozeduraufruf (@PASS) wird die Parameterliste dort abgelegt.

Die Angabe, ob eine Übergabe in STANDARD- oder OPTIMAL-Form stattfinden soll, erfolgt

- in der aufrufenden Prozedur im PASS-Operanden des Prozeduraufrufs (@PASS),
- in der aufgerufenen Prozedur im PASS-Operanden des Prozedurkopfes (@ENTR).

Beide Angaben müssen übereinstimmen.

Im Programm "DEMOPARA" (siehe Anhang 11.5) sind die verschiedenen Möglichkeiten der Parameterübergabe und -übernahme in einem Beispiel dargestellt.

Die folgende Tabelle zeigt die Kombinationsmöglichkeiten der Formen der Parameterübergabe mit denen der Parameterübernahme.

	Parameterübernahme,			
Parameterübergabe,	STANDARD-Schnittst. (9.5.3.1)	OPTIMAL-Schnittst. (9.5.3.2)	in Formal-parameter (9.5.3.3)	in Formalpar. im LOCAL (9.5.3.4)
STANDARD-Schnittst., statisch (9.5.1.1)	X	-	X	X
STANDARD-Schnittst., dynamisch (9.5.1.2)	X	-	X	X
OPTIMAL-Schnittst., dynamisch (9.5.2)	-	X	X	X

Tabelle 9-3 Kombinationen von Parameterübergabe und Parameterübernahme

9.5.1 Parameterübergabe über die STANDARD-Schnittstelle

Bei der Parameterübergabe über die STANDARD-Schnittstelle werden die Parameter in einer Parameterliste zusammengefaßt. Die Adresse der Parameterliste wird in Register 1 übergeben.

9.5.1.1 Statische Parameterübergabe

Bei der statischen Parameterübergabe wird die Parameterliste mit konstanten Werten zur Übersetzungszeit erzeugt. Sie bleibt bis zum Programmende bestehen. Die Parameterliste wird über den @PAR-Makro (Format 1) innerhalb der rufenden Prozedur gespeichert, d.h., sie gehört zum entsprechenden Modul.

Diese Form der Parameterübergabe ist erlaubt in Prozeduren vom Typ M, E, I, L und B.

In der rufenden Prozedur

- gibt der PAR-Operand von @PASS (Format 2) den Namen der zu übergebenden Parameterliste an. Register 1 wird mit dieser Adresse geladen.
- muß die Parameterliste mit @PAR (Format 1) zwischen @EXIT und @END erzeugt werden. Der Name von @PAR gibt den Namen der Parameterliste an.
- vergibt der PLIST-Operand von @PAR (Format 1) Namen an die Adreßkonstanten der Parameterliste. Fehlt der PLIST-Operand, werden namenlose Konstanten erzeugt.
- enthält der VLIST-Operand von @PAR (Format 1) die Aktualparameter. Dies sind entweder der Name eines Feldes, dessen Adresse übergeben werden soll, oder ein selbstdefinierender Wert.

  Ist ein Aktualparameter nicht vorhanden, muß im VLIST-Operanden ein zusätzliches Komma geschrieben werden.

In der gerufenenen Prozedur

gibt es für die Übernahme der Parameter die folgenden Möglichkeiten:

- Ansprechen der Parameter über die Adresse der Parameterliste (STANDARD-Schnittstelle, siehe 9.5.3.1),
- Übernahme der Parameter in Formalparameter, denen Felder in der gerufenen Prozedur entsprechen (siehe 9.5.3.3) und
- Übernahme der Parameter in Formalparameter, denen Felder im LOCAL-Bereich der gerufenen Prozedur entsprechen (siehe 9.5.3.4).

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-------------------------------------------
* rufende Prozedur
PRO1        | @ENTR     | TYP=M
            | @DATA     | CLASS=S,BASE=R6,INIT=PRO1DAT
            | .         |
            | .         |
            | @PASS     | NAME=PRO2,PAR=PARLIST
            | .         |
            | .         |
            | @EXIT     |
PARLIST     | @PAR      | PLIST=(PAR1,PAR2,PAR3),VLIST=(FIELD,27,SET)
            | @END      |
PRO1DAT     | DS        | 0H
FIELD       | DC        | C'ABC'
SET         | DC        | C'DEF'
            | .         |
            | .         |
```

Die Prozedur PRO1 ruft PRO2 auf und übergibt drei Parameter.
Übergeben wird die Adresse von FIELD, der Wert 27 und die Adresse von SET.

```
* gerufene Prozedur
PRO2        | @ENTR     | TYP=I,LOCAL=IN,PLIST=(IN1,IN2,IN3)
            | @DATA     | CLASS=S,BASE=R7,INIT=PRO2DAT
            | .         |
            | .         |
            | L         | R8,IN3
            | MVC       | FIELDB,0(R8)
            | .         |
            | .         |
            | @EXIT     |
            | @END      |
IN          | @PAR      | D=YES,LEND=YES,PLIST=(IN1,IN2,IN3)
            | .         |
            | .         |
PRO2DAT     | DS        | 0H
FIELDA      | DS        | CL3
FIELDB      | DS        | CL3
            | .         |
            | .         |
```

PRO2 übernimmt drei Parameter in ihren LOCAL-Bereich und überträgt den Inhalt von SET nach FIELDB.

9.5.1.2 Dynamische Parameterübergabe

Bei der dynamischen Parameterübergabe wird die Parameterliste zur Laufzeit erzeugt und im Sicherstellungsbereich der rufenden Prozedur abgelegt. Bei der Rückkehr aus dieser Prozedur wird der Speicherbereich wieder freigegeben. Der Aufbau der Parameterliste erfolgt über den PLIST-Operanden von @PASS (Format 2) in der rufenden Prozedur.

Dynamische Parameterübergabe ist nur möglich in Prozeduren der Typen M, E und I.

In der rufenden Prozedur

- gibt der MAXPRM-Operand von @ENTR (Format 1 und 2) an, wieviele Parameter die Prozedur maximal übergeben darf. Entsprechend dieser Angabe wird im LOCAL-Bereich der rufenden Prozedur Speicherplatz für die Maximalgröße der Parameterliste reserviert.
- gibt der PLIST-Operand von @PASS (Format 3) die Namen der Felder, bzw. die Register an, deren Adresse an die rufende Prozedur übergeben werden sollen (Aktualparameter).
- gibt der PASS-Operand von @PASS (Format 3) an, daß die Übergabe über die STANDARD-Schnittstelle erfolgen soll.

In der gerufenen Prozedur

gibt es für die Übernahme der Parameter die folgenden Möglichkeiten:

- Ansprechen der Parameter über die Adresse der Parameterliste (STANDARD-Schnittstelle, siehe 9.5.3.1),
- Übernahme der Parameter in Formalparameter, denen Felder in der gerufenen Prozedur entsprechen (siehe 9.5.3.3) und
- Übernahme der Parameter in Formalparameter, denen Felder im LOCAL-Bereich der gerufenen Prozedur entsprechen (siehe 9.5.3.4).

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-----------------------------------------------
* rufende Prozedur
PRO1        | @ENTR     | TYP=M,MAXPRM=2
            | @DATA     | CLASS=A,BASE=R6,DSECT=DUMMY,INIT=PRO1DAT
            | .         |
            | .         |
            | @PASS     | NAME=PRO2,PLIST=(D1,D2),PASS=STA
            | .         |
            | .         |
            | @EXIT     |
PRO1DAT     | DS        | 0H
FIELD       | DC        | C'ABC'
SET         | DC        | C'DEF'
            | @END      |
DUMMY       | DSECT     |
D1          | DS        | CL3
D2          | DS        | CL3
            | .         |
            | .         |
PRO1        | CSECT     |
```

Die Prozedur PRO1 ruft PRO2 auf und übergibt zwei Parameter. Die Felder, deren Adressen übergeben werden sollen, sind in DUMMY definiert und werden mit den Werten aus PRO1DAT initialisiert.

```
* gerufene Prozedur
PRO2        | @ENTR     | TYP=I,LOCAL=IN,PLIST=(IN1,IN2),PASS=STA
            | @DATA     | CLASS=S,BASE=R7,INIT=PRO2DAT
            | .         |
            | .         |
            | L         | R8,IN2
            | MVC       | FIELDB,0(R8)
            | .         |
            | .         |
            | @EXIT     |
            | @END      |
IN          | @PAR      | D=YES,LEND=YES,PLIST=(IN1,IN2)
            | .         |
            | .         |
PRO2DAT     | DS        | 0H
FIELDA      | DS        | CL3
FIELDB      | DS        | CL3
            | .         |
            | .         |
```

PRO2 übernimmt zwei Parameter in ihren LOCAL-Bereich und überträgt den Inhalt von D2 nach FIELDB.

9.5.2 Parameterübergabe über die OPTIMAL-Schnittstelle

Die Parameterübergabe über die OPTIMAL-Schnittstelle ist nur dynamisch möglich. Hier werden die Parameter in den Registern 1 bis 4 direkt übergeben. In diesen Registern werden die Adressen der Felder, bzw. die Werte abgelegt, die an die gerufene Prozedur übergeben werden sollen.

Bei mehr als vier Parametern werden die ersten drei Parameter in den Registern 2 bis 4 übergeben. Für die restlichen Parameter wird eine Parameterliste angelegt, deren Adresse in Register 1 geladen wird. D.h., drei Parameter werden über die OPTIMAL-Schnittstelle übergeben, die weiteren über die STANDARD-Schnittstelle.

Dynamische Parameterübergabe ist nur möglich in Prozeduren der Typen M, E und I.

Bei der Verknüpfung mit Programmen in anderen Programmiersprachen ist diese Form der Parameterübergabe **nicht** zulässig.

In der rufenden Prozedur

- gibt der PLIST-Operand von @PASS (Format 3) die Aktualparameter an, d.h., die Adressen der Felder, bzw. die Register, die an die gerufene Prozedur übergeben werden sollen.
- gibt der PASS-Operand von @PASS (Format 3) an, daß die Übergabe über die OPTIMAL-Schnittstelle erfolgen soll.

In der gerufenen Prozedur

gibt es für die Übernahme der Parameter die folgenden Möglichkeiten:

- Direktes Ansprechen der Parameter über die Register 1 bis 4 (OPTIMAL-Schnittstelle, siehe 9.5.3.2),
- Übernahme der Parameter in Formalparameter, denen Felder in der gerufenen Prozedur entsprechen (siehe 9.5.3.3) und
- Übernahme der Parameter in Formalparameter, denen Felder im LOCAL-Bereich der gerufenen Prozedur entsprechen (siehe 9.5.3.4).

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-----------------------------------------------
* rufende Prozedur
PRO1        | @ENTR     | TYP=M
            | @DATA     | CLASS=A,BASE=R6,DSECT=DUMMY,INIT=PRO1DAT
            | .         |
            | .         |
            | @PASS     | NAME=PRO2,PLIST=(D1,D2),PASS=OPT
            | .         |
            | .         |
            | @EXIT     |
PRO1DAT     | DS        | 0H
FIELD       | DC        | C'ABC'
SET         | DC        | C'DEF'
            | @END      |
DUMMY       | DSECT     |
D1          | DS        | CL3
D2          | DS        | CL3
            | .         |
            | .         |
PRO1        | CSECT     |
```

Die Prozedur PRO1 ruft PRO2 auf und übergibt zwei Parameter. Die Felder, deren Adressen übergeben werden sollen, sind in DUMMY definiert und werden mit den Werten aus PRO1DAT initialisiert.

```
* gerufene Prozedur
PRO2        | @ENTR     | TYP=I,LOCAL=IN,PLIST=(IN1,IN2),PASS=OPT
            | @DATA     | CLASS=S,BASE=R7,INIT=PRO2DAT
            | .         |
            | .         |
            | L         | R8,IN2
            | MVC       | FIELDB,0(R8)
            | .         |
            | .         |
            | @EXIT     |
            | @END      |
IN          | @PAR      | D=YES,LEND=YES,PLIST=(IN1,IN2)
            | .         |
            | .         |
PRO2DAT     | DS        | 0H
FIELDA      | DS        | CL3
FIELDB      | DS        | CL3
            | .         |
            | .         |
```

PRO2 übernimmt zwei Parameter in ihren LOCAL-Bereich und überträgt den Inhalt von D2 nach FIELDB.

9.5.3 Parameterübernahme

Für die Übernahme von Parametern gibt es, abhängig von der Übergabeform und dem Prozedurtyp, verschiedene Möglichkeiten.

9.5.3.1 Parameterübernahme über die STANDARD-Schnittstelle

Bei der STANDARD-Übergabe enthält Register 1 immer die Adresse der Parameterliste. Die gerufene Prozedur kann auf diese STANDARD-Schnittstelle aufsetzen und die Parameter über die Adresse der Parameterliste ansprechen.

Diese Form ist in allen Prozedurtypen erlaubt.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------------------
* rufende Prozedur
PRO1        | @ENTR     | TYP=M
            | @DATA     | CLASS=S,BASE=R6,INIT=PRO1DAT
            | .         |
            | .         |
            | @PASS     | NAME=PRO2,PAR=PARLIST
            | .         |
            | .         |
            | @EXIT     |
PARLIST     | @PAR      | PLIST=(PAR1,PAR2,PAR3),VLIST=(FIELD,27,SET)
            | @END      |
PRO1DAT     | DS        | 0H
FIELD       | DC        | C'ABC'
SET         | DC        | C'DEF'
            | .         |
            | .         |
```

Die Prozedur PRO1 ruft PRO2 auf und übergibt drei Parameter (statische Übergabe). Übergeben wird die Adresse von FIELD, der Wert 27 und die Adresse von SET.

```
* gerufene Prozedur
PRO2        @ENTR       TYP=I
            @DATA       CLASS=S,BASE=R7,INIT=PRO2DAT
            .
            .
            L           R8,0(0,R1)
            MVC         FIELDA,0(R8)
            L           R8,8(0,R1)
            MVC         FIELDB,0(R8)
            .
            .
            @EXIT
            @END
            .
            .
PRO2DAT     DS          0H
FIELDA      DS          CL3
FIELDB      DS          CL3
            .
            .
```

PRO2 übernimmt die Parameter und überträgt den Inhalt von FIELD nach FIELDA und den Inhalt von SET nach FIELDB.

9.5.3.2 Parameterübernahme über die OPTIMAL-Schnittstelle

Bei der OPTIMAL-Übergabe werden die Parameter in den Registern 1 bis 4 übergeben. Die gerufene Prozedur kann direkt auf diese OPTIMAL-Schnittstelle aufsetzen und die Parameter über die Register 1 bis 4 ansprechen. Bei mehr als vier Parametern muß die gerufene Prozedur auch die entsprechende Verwaltung übernehmen.

In der gerufenen Prozedur muß im @ENTR-Makro (Format 2) der PASS-Operand angeben, daß die Übergabe über die OPTIMAL-Schnittstelle stattfindet.

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-----------------------------------------------
* rufende Prozedur
PRO1        | @ENTR     | TYP=M
            | @DATA     | CLASS=A,BASE=R6,DSECT=DUMMY,INIT=PRO1DAT
            | .         |
            | .         |
            | @PASS     | NAME=PRO2,PLIST=(D1,D2),PASS=OPT
            | .         |
            | .         |
            | @EXIT     |
PRO1DAT     | DS        | 0H
FIELD       | DC        | C'ABC'
SET         | DC        | C'DEF'
            | @END      |
DUMMY       | DSECT     |
D1          | DS        | CL3
D2          | DS        | CL3
            | .         |
            | .         |
PRO1        | CSECT     |
```

Die Prozedur PRO1 ruft PRO2 auf und übergibt zwei Parameter. Die Felder, deren Adressen übergeben werden sollen, sind in DUMMY definiert und werden mit den Werten aus PRO1DAT initialisiert.

```
* gerufene Prozedur
PRO2        @ENTR       TYP=I,PASS=OPT
            @DATA       CLASS=S,BASE=R7,INIT=PRO2DAT
            .
            .
            MVC         FIELDB,0(R2)
            .
            .
            @EXIT
            @END
            .
            .
PRO2DAT     DS          0H
FIELDA      DS          CL3
FIELDB      DS          CL3
            .
            .
```

PRO2 übernimmt zwei Parameter und überträgt den Inhalt von D2 nach FIELDB.

9.5.3.3 Parameterübernahme in Formalparameter

Den Formalparametern entsprechen in diesem Fall Felder, die in der gerufenen Prozedur definiert sind.

Diese Form der Parameterübernahme ist nicht sinnvoll für Prozeduren vom Typ B oder L. Außerdem sollte sie nicht angewendet werden für READ-ONLY-Prozeduren und bei rekursiven Prozeduraufrufen.

In der gerufenen Prozedur

- muß der PLIST-Operand von @ENTR (Format 2) die Liste der Formalparameter enthalten. In diese Datenfelder werden die Einträge aus der Parameterliste übernommen.
- muß der PASS-Operand von @ENTR (Format 2) angeben, ob die Übergabe der Parameter über die OPTIMAL- oder die STANDARD-Schnittstelle stattfindet.
- müssen die Datenfelder, die im PLIST-Operanden angegeben sind, vom Anwender in der Prozedur definiert werden. Dies ist nur möglich mit Hilfe von @DATA CLASS=S.

Beispiel

```
Name       | Operation | Operanden
-----------------------------------------------------------------
* rufende Prozedur
PRO1       | @ENTR     | TYP=M
           | @DATA     | CLASS=S,BASE=R6,INIT=PRO1DAT
           | .         |
           | .         |
           | @PASS     | NAME=PRO2,PAR=PARLIST
           | .         |
           | .         |
           | @EXIT     |
PARLIST    | @PAR      | PLIST=(PAR1,PAR2,PAR3),VLIST=(FIELD,27,SET)
           | @END      |
PRO1DAT    | DS        | 0H
FIELD      | DC        | C'ABC'
SET        | DC        | C'DEF'
           | .         |
           | .         |
```

Die Prozedur PRO1 ruft PRO2 auf und übergibt drei Parameter (statische Übergabe). Übergeben wird die Adresse von FIELD, der Wert 27 und die Adresse von SET.

```
* gerufene Prozedur
PRO2        @ENTR       TYP=I,PLIST=(IN1,IN2,IN3)
            @DATA       CLASS=S,BASE=R7,INIT=PRO2DAT
            .
            .
            L           R8,IN3
            MVC         FIELDB,0(R8)
            .
            .
            @EXIT
            @END
            .
            .
PRO2DAT     DS          0H
FIELDA      DS          CL3
FIELDB      DS          CL3
IN1         DS          F
IN2         DS          F
IN3         DS          F
```

PRO2 übernimmt drei Parameter und überträgt den Inhalt von SET nach FIELDB.

9.5.3.4 Parameterübernahme in Formalparameter im LOCAL-Bereich

Den Formalparametern entsprechen in diesem Fall Felder, die im LOCAL-Bereich des Prozedur-STACKS (siehe 9.4.2) der gerufenen Prozedur liegen. Die Struktur dieses Bereiches beschreibt ein Pseudoabschnitt, der mit der Struktur der Parameterliste übereinstimmen muß. In der gerufenen Prozedur bezeichnet dann ein Formalparameter, der im Pseudoabschnitt definiert wurde, bzw. das entsprechende Register, die Adresse der Konstanten aus der rufenden Prozedur.

Diese Form der Parameterübernahme ist nur erlaubt für Prozeduren vom Typ E oder I. Sie sollte unbedingt angewendet werden für READ-ONLY-Prozeduren und bei rekursiven Prozeduraufrufen.

In der gerufenen Prozedur

- muß mit dem LOCAL-Operanden von @ENTR (Format 2) ein LOCAL-Bereich im Prozedur-STACK angefordert werden, in den die Parameter übernommen werden sollen.
- muß der PLIST-Operand von @ENTR (Format 2) die Liste der Formalparameter enthalten. In diese Datenfelder, bzw. Register werden die Einträge aus der Parameterliste übernommen.
- muß der PASS-Operand von @ENTR (Format 2) angeben, ob die Übergabe der Parameter über die OPTIMAL- oder die STANDARD-Schnittstelle stattfindet.
- muß mit @PAR (Format 2) unmittelbar nach der Prozedur ein Pseudoabschnitt generiert werden. Der Name dieses Pseudoabschnitts muß mit dem Namen im LOCAL-Operanden von @ENTR übereinstimmen.
- gibt der PLIST-Operand von @PAR die Namen der Formalparameter im Pseudoabschnitt an, die die Struktur des LOCAL-Bereichs beschreiben. Die PLIST-Operanden von @PAR müssen mit den PLIST-Operanden von @ENTR übereinstimmen.

Beispiel siehe 9.5.1, Statische Parameterübergabe oder 9.5.2, Dynamische Parameterübergabe

9.6 ILCS-Anschluß für die Strukturierte Programmierung

Mit der Version V1.1A des ASSEMBH sind die Makros zur Strukturierten Programmierung (@-Makros, siehe Kapitel 10) und das Assembler-Laufzeitsystem (ASSEMBH-RTS) dahingehend erweitert, daß der Anwender auch in Assembler ILCS-fähige Programme schreiben kann.
Folgende Leistungen werden in der ILCS-Umgebung bereitgestellt:

Durch neue Operanden im @ENTR-Makro für die Hauptprozedur werden folgende ILCS-Funktionen ermöglicht:

- Anmeldung benutzereigener Routinen zur Speicherbeschaffung und Speicherfreigabe für Stack- und Heap-Speicher
- Anmeldung benutzereigener Beendigungs-Routinen
- Angabe der minimalen Stack-Extent-Größe

Über diese Anpassungen an die ILCS-Norm hinaus wird dem Anwender mit neuen Strukturmakros der Zugang zu folgenden prozedurübergreifenden Funktionen von ILCS ermöglicht:

- Behandlung von Ereignissen (Event's)
- Behandlung von Folgeprozessen (Contingencies)
- Behandlung von Unterbrechungen (STXIT's)

Die neuen Makros sollen in ILCS-Prozeduren notwendige Aufrufe von Systemmakros für die genannten Behandlungsarten ersetzen.

Durch zusätzliche neue Makros werden folgende Funktionen ermöglicht:

- Behandlung von Programmasken
- Setzen der Monitorjobvariablen
- Sprachinitialisierung bei nachgeladenen Moduln

Die neuen @-Makros generieren den Aufruf entsprechender Eingänge im Standard-Event-Handler (SEH), im Standard-Contingency-Handler (SCH), sowie im Standard-STXIT-Handler (SSH) der ILCS-Schnittstelle.

Will der Anwender keine ILCS-Prozedur schreiben, so kann er die Makros auf die gleiche Weise wie bisher aufrufen. Für bestehende Quellen, die nicht auf ILCS umgestellt werden sollen, sind deshalb weder Source-Änderungen noch eine neue Übersetzung notwendig.

9.6.1 Prozedurverknüpfung

Durch die Erweiterungen der Makros @ENTR, @PASS, @PAR und @EXIT kann die Prozedurschnittstelle auf die ILCS-Konventionen umgestellt werden.

Dies betrifft den Prozedurprolog, den Prozedurepilog mit Rückgabe von Funktionswert und Returncode sowie den Prozeduraufruf mit Übergabe von Parametern.

Registerkonventionen (ILCS- und Nicht-ILCS-Schnittstelle)

Die folgende Übersicht enthält einen Vergleich der Konventionen der ILCS- und Nicht-ILCS-Schnittstelle für

- Registerverwendung bei Prozeduraufruf mit/ohne Parameter
- Registerrestaurierung bei Prozedurende mit/ohne Funktionswert und Returncode
- Longjump über mehrere Prozedurebenen bei Prozedurende.

Registerverwendung bei Prozeduraufruf

ILCS	Nicht-ILCS
R15 enthält die Adresse der gerufenen Prozedur	wie ILCS
R14 enthält die Rückkehr-adresse	wie ILCS
R1 enthält die Adresse der Parameterliste bei Parameterversorgung STANDARD; im letzten Wort der Liste ist linkes Bit gesetzt	wie ILCS
R0 enthält die Anzahl der übergebenen Aktualparameter	Parameteranzahl steht nicht in R0; im letzten Wort der Liste ist linkes Bit gesetzt
Parameterübergabe OPTIMAL ist nicht erlaubt	Übergabe OPTIMAL neben STANDARD zulässig
Parameterübergabe "call by reference", d.h. R1 enthält die Adresse einer Parameter-adreßliste	Sowohl die Parameterübergabe "call by value", als auch die Übergabe "call by reference" sind möglich.

Registersicherung bei Prozedurende

ILCS	Nicht-ILCS
R2 bis R14 werden restauriert	keine Angabe bei @EXIT/@ENTR: R2 bis R14 restauriert R0, R1: Funktionswerte R15: Returncode
RETURNS=YES (@ENTR): R0 und R1: Funktionswert	RETURNS=YES (@ENTR): R2 bis R14 und R0 restauriert R1: Funktionswert R15: Returncode
RETURNS=NO (@ENTR): R0 und R1: undefiniert	RETURNS=NO (@ENTR): R0 bis R14 restauriert R15: Returncode
	RESTORE=MIN (@EXIT): R7 bis R14 restauriert R0 bis R6: Funktionswerte R15: Returncode
	PROG=FORTRAN (@EXIT): R2 bis R14 restauriert R1: von FORTRAN nicht ausgewertet R0: Returncode

Longjump

ILCS	Nicht-ILCS
Nicht zulässig	Über @EXIT TO = möglich, nur in Nicht-ILCS-Umgebung ablauffähig

Parameterübergabe

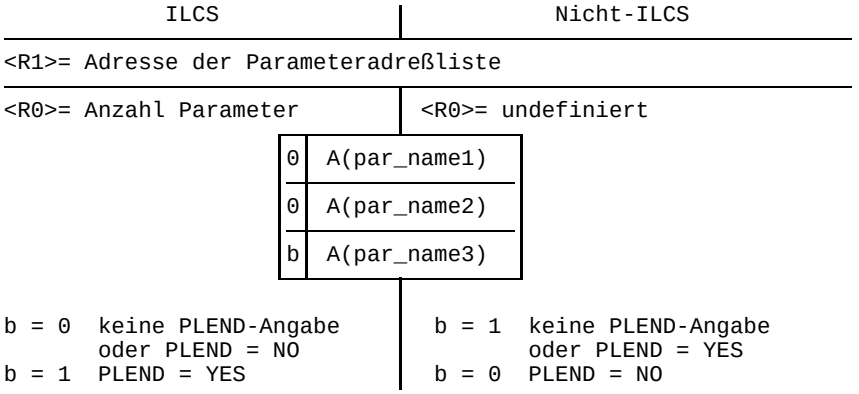
Es gibt zwei Möglichkeiten der Parameterübergabe und -übernahme:
- in STANDARD-Form
- durch "call by reference"
  Der Anwender muß in der Aktualparameterliste (@PAR-/@PASS-Makro) nicht die Werte, sondern die Adressen der zu übergebenden Parameterwerte angeben.

Beachte:
sowohl bei statischer als auch bei dynamischer Parameterübergabe wird in der Parameterliste im letzten Parameter das linke Bit, abhängig vom Parameter PLEND gesetzt.

Verboten sind:
- bei @ENTR: der Parameter PASS=OPTIMAL
- bei @PASS: der Parameter PASS=OPTIMAL und bei der statischen Parameterübergabe der Parameter PAR=(<register>).
- bei @EXIT: die Parameter RESTORE=MIN, TO=<proz.name> und PROG=FORTRAN.

Beispiele für Parameterübergabe

1. ’Call by reference’
   @PASS ziel, PLIST=(par_name1, par_name2, par_reg), PASS=STANDARD wobei <par_reg>=par_name3



2. ’Call by reference’ und ’Call by value’
   @PASS aufrufziel, PLIST=(par_name1,2, par_name3), PASS=STANDARD

ILCS	Nicht-ILCS
Direktwert nur erlaubt, wenn die rufende Prozedur diesen als absolute A d r e s s e interpretiert	Direktwert immer erlaubt, wird als absolute Adresse oder als Direktwert interpretiert
<R1>= Adresse der Parameteradreßliste	
<R0>= Anzahl Parameter	<R0>= undefiniert

0	A(par_name1)
0	A(2)
b	A(par_name3)

b = 0 keine PLEND-Angabe oder PLEND = NO b = 1 PLEND = YES	b = 1 keine PLEND-Angabe oder PLEND = YES b = 0 PLEND = NO

3. ’Call by reference’ und fehlender Parameter
   @PASS aufrufziel, PLIST=(par_name1,, par_name3), PASS=STANDARD

ILCS	Nicht-ILCS
Nicht erlaubt MNOTE (Significant Error)	Nicht erlaubt Mnote und Flag (Severe)
<R1>= undefiniert	<R1>= Adresse der Parameter-adreßliste
<R0>= undefiniert	

0	A(par_name1)
0	A(0)
1	A(par_name3)

9.6.2 Benutzereigene Routinen anmelden

Mit ILCS kann der Anwender bei der Initialisierung benutzereigene Routinen anmelden:
- für Speicherbeschaffung und Speicherfreigabe für Stack- und Heap-Speicher
- zur Beendigungs-Behandlung
- zur Festlegung der minimalen Stack-Extent-Größe

Diese Routinen müssen die ILCS-Konventionen einhalten. Sie gelten für die gesamte ILCS-Umgebung und können nur bei der Hauptprozedur @ENTR mit TYP = M definiert werden (siehe @ENTR, Operand STREQ ff; Kapitel 10).

9.6.3 Ereignisse behandeln (Event Handling)

ILCS stellt mit dem Standard-Event-Handler (SEH) eine Routine zur Verfügung, die in ILCS-Programmen auftretende Ereignisse koordiniert.

An Ereignissen werden von ILCS behandelt:
- STXIT-Ereignisse im Sinne der BS2000-STXIT-Klassen PROCHK und ERROR.
- Nicht-STXIT-Ereignisse, die vom BS2000 nicht abgedeckt werden (z.B. OPEN-Fehler bei Dateizugriff).

STXIT-Ereignisse werden vom System zugestellt, während Nicht-STXIT-Ereignisse von einer Prozedur dem Standard-Event-Handler signalisiert werden müssen.

Die Ereignisbehandlungsroutinen für die verschiedenen Ereignisklassen werden durch den @ENTR-Makro mit Angabe von entsprechenden Parametern bereitgestellt.

Das dynamische Anmelden der vom Benutzer im Prozedurprolog angegebenen Ereignisbehandlungsroutinen erfolgt bei Aufruf einer Prozedur, das Abmelden beim Verlassen einer Prozedur.

Ein explizites An- und Abmelden dieser Routinen durch den Benutzer entfällt.

Der Standard-Event-Handler bekommt immer dann die Kontrolle, wenn vom Benutzer für ein aufgetretenes Ereignis keine STXIT-Routine beim Standard-STXIT-Handler angemeldet wurde bzw. wenn der Standard-Event-Handler nicht beendet wurde.

Für jeden durch ein Ereignis ausgelösten STXIT- oder Contingency-Prozeß wird von ILCS ein separater Stack und Heap beschafft und eingerichtet. Bei einem STXIT-Ereignis im Sinne des BS2000 wird vom Standard-Event-Handler der Zustand der Unterbrechungsstelle festgehalten. Für Nicht-STXIT-Ereignisse übergibt der Benutzer beim Signalisieren dieses Ereignisses an den Standard-Event-Handler einen Parameterblock, der die Beschreibung der Unterbrechungsstelle enthält.

Der Standard-Event-Handler sucht ab der aktuellen Save-Area innnerhalb der Save-Area-Verkettung in einer jeweils angeschlossenen Event-Handler-List nach der Adresse einer für dieses Ereignis zuständigen Behandlungsroutine. Ist eine solche gefunden, so wird sie vom Standard-Event-Handler nach ILCS-Konventionen gerufen; als Parameter wird vom Standard-Event-Handler der Kontext der Unterbrechungsstelle, erweitert um einen zusätzlichen Parameterblock, an die Routine übergeben.

Im einzelnen können folgende Funktionen des Standard-Event-Handlers über neue Strukturmakros aufgerufen werden:

- Ereignisbehandlungsroutinen anmelden (siehe @ENTR, Operanden ABKR, PROCHK, ERROR und OTHEVT; Kapitel 10)
- Nicht-STXIT-Ereignis signalisieren (siehe @EVTOE, Kapitel 10)

9.6.4 Folgeprozesse behandeln (Contingency Handling)

Neben der Ereignisbehandlung bietet ILCS mit dem Standard-Contingency-Handler (SCH) eine Routine an, die dem Benutzer die Möglichkeit gibt, in ILCS-Programmen mit Folgeprozessen zu arbeiten.

Diese Folgeprozesse sind vom Benutzer zu definierende externe Prozeduren, die vom Standard-Contingency-Handler verwaltet und nach ILCS-Konventionen aufgerufen werden.

Mit Hilfe neuer Strukturmakros kann eine Contingency-Routine an- und abgemeldet werden.

Das Eintreten eines Ereignisses muß vom Benutzer weiterhin mit dem Systemmakro POSSIG (Sende Signal) für eine Ereigniskennung signalisiert werden. Die Anforderung für ein Signal muß mit dem Systemmakro SOLSIG erfolgen.

Ein auftretendes Ereignis, etwa ein Signal an eine Ereigniskennung, aktiviert zunächst eine der Benutzerroutine zugeordnete Standard-Contingency-Handler-Routine, die für die Behandlungsroutine die notwendige Umgebung einrichtet: dazu gehört ein separater Heap sowie ein Stack für die Save-Areas des Benutzerprozesses.

Danach ruft der Standard-Contingency-Handler die Benutzer-Contingency-Routine nach ILCS-Konventionen auf.

Folgende Einzelfunktionen können über die neue Makroschnittstelle erreicht werden:

- Contingency-Routine anmelden (siehe @CONEN, Kapitel 10)
- Contingency-Routine abmelden (siehe @CONDI, Kapitel 10)

9.6.5 Unterbrechungsereignisse behandeln (STXIT Handling)

Der Benutzer kann innerhalb von ILCS-Programmen auf Prozedurebene mit eigenen STXIT-Routinen arbeiten. Dafür stellt ILCS mit dem Standard-STXIT-Handler (SSH) eine Routine zur Verfügung, die bei auftretenden STXIT-Ereignissen vom Benutzer angemeldete Behandlungsroutinen nach ILCS-Konventionen aufruft.

Das An- und Abmelden dieser Routinen in ILCS-Umgebung übernehmen neue Strukturmakros mit der entsprechenden Parameterversorgung.

Von ILCS werden alle im BS2000 definierten STXIT-Ereignisse unterstützt.

In strukturierten ILCS-Programmen können die expliziten Aufrufe von BS2000-Makros für STXIT-Behandlung durch die neuen Strukturmakros ersetzt werden. Sie sorgen in ILCS-Umgebung für den ILCS-konformen Aufruf des STXIT-Handlers.

Der Standard-STXIT-Handler bekommt immer dann die Kontrolle, wenn für ein STXIT-Ereignis Benutzer-STXIT-Routinen angemeldet wurden; für einen STXIT-Prozeß wird ein separater Stack und Heap eingerichtet.

Die angemeldeten STXIT-Routinen werden vom Standard-STXIT-Handler gemäß der definierten BS2000-Bearbeitungsreihenfolge nach ILCS-Konventionen aufgerufen.

Wenn alle STXIT-Routinen abgearbeitet sind, gibt der Standard-STXIT-Handler den separaten Stack und Heap frei und richtet wieder die Umgebung der unterbrochenen Prozedur ein.

Folgende ILCS-Funktionen sind mit diesen Makros erreichbar:

- STXIT-Routine anmelden (siehe @STXEN, Kapitel 10)
- STXIT-Routine abmelden (siehe @STXDI, Kapitel 10)

9.6.6 Programmaske setzen

Mit dem neuen Strukturmakro @SETPM (siehe Kapitel 10) kann der Benutzer in Prozeduren die Programmaske einschließlich der Anzeige dynamisch so verändern, daß bei bestimmten mathematischen Operationen auftretende Ereignisse vom System nicht auf Programmunterbrechung geleitet werden.

Solche Ereignisse können sein:

- Festpunktüberlauf
- Dezimalüberlauf
- Exponentenunterlauf
- Mantisse=0

Der Makro ersetzt in strukturierten ILCS-Programmen den BS2000-Befehl zum Setzen der Programmaske.

9.6.7 MONJV-Wert in der PCD setzen

Mit dem neuen Strukturmakro @SETJV (siehe Kapitel 10) kann der Benutzer in Prozeduren einen Wert definieren, der in das entsprechende Feld der PCD (siehe ASSEMBH, Benutzerhandbuch [1], 'ILCS-Datenstrukturen') eingetragen wird. Mit diesem MONJV-Wert als Parameter wird von ILCS der Systemmakro TERM aufgerufen.

9.6.8 Sprachinitialisierung bei nachgeladenen Moduln

Innerhalb von strukturierten ILCS-Programmen können Moduln nachgeladen werden. Mit dem Makro @ININ (siehe Kapitel 10) muß dann ein Aufruf von ILCS erfolgen:
ILCS prüft, ob eine Sprachinitialisierung notwendig ist. Ist dies der Fall, so aktiviert ILCS die Initialisierungs-Routine, wenn für sie noch kein Aufruf stattgefunden hat.

10 Vordefinierte Makros für die Strukturierte Programmierung

Allgemeine Programmierhinweise

Format der Makroaufrufe

Die vordefinierten Makros für die Strukturierte Programmierung werden über ihren jeweiligen Makroaufruf angesprochen. Das erste Zeichen dieser Makroaufrufe ist ein "@", deshalb werden diese Makros auch "@-Makros" genannt.

Das Format der vordefinierten Makroaufrufe entspricht dem Format für Makroaufrufe, wie es in 7.1 ausführlich beschrieben ist. Die dort aufgeführten Regeln gelten hier ebenso.

Namens-, Operations- und Operandeneintrag der vordefinierten Makroaufrufe können bei entsprechenden Voraussetzungen mit Hilfe von **variablen Parametern** generiert werden (siehe Kap. 6, variable Parameter).

Die Angabe eines Folgesymbols im Namenseintrag ist bei einem Makroaufruf im allgemeinen möglich. Im Rahmen der strukturierten Programmierung sollte dies jedoch unbedingt vermieden werden.

Operanden der Makroaufrufe

Stellungsoperanden werden auf Grund ihrer Stellung im Operandeneintrag zugewiesen. Ihre Reihenfolge muß daher der im Format angegebenen entsprechen.

Kennwortoperanden sind durch das Gleichheitszeichen (=) kenntlich gemacht. Ihnen wird im Makroaufruf über das Kennwort ein Wert zugewiesen. Die Reihenfolge der Kennwortoperanden im Makroaufruf ist beliebig (siehe auch 7.1.1, Kennwort- und Stellungsoperanden).

Namen

Bei der strukturierten Programmierung gibt es für Register vordefinierte Namen. Die Mehrzweckregister können ohne vorherige explizite Zuweisung mit den Namen R0, R1, ..., R15 angesprochen werden, die Gleitpunktregister mit FA, FB, FC und FD.

Zusätzlich zu diesen vordefinierten Namen sind alle Namen, die mit "@" oder mit "R@" beginnen, für den ASSEMBH reserviert und dürfen nicht anderweitig verwendet werden.

@AND Logisches 'Und'

Funktion

@AND realisiert die logische Und-Verknüpfung in zusammengesetzten Bedingungen.

Format

Name	Operation	Operanden
[name]	@AND	cond_sym

name Name
cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe 9.2.4 und 9.2.5)

Programmierhinweis

Dem Aufruf des @AND-Makro muß grundsätzlich ein Anzeige-setzender Assemblerbefehl folgen (siehe Assemblerbefehle [3]).

Beispiel

```
Name        Operation   Operanden
            .
            .
            @IF         ZE
            C           R4,ZERO          (01)
            @AND        NZ
            LTR         R5,R5            (02)
            @THEN
            ST          R1,POINT
            @BEND
            .
```

Der @THEN-Zweig wird nur ausgeführt, wenn die *erste Bedingung* (01) und die *zweite Bedingung* (02) zutrifft.

>>>>> siehe auch @OR und @TOR

@BEGI Sequenz

Funktion

@BEGI bildet den Eintrittspunkt eines Sequenz-Strukturblocks.

Format

Name	Operation	Operanden
[name]	@BEGI[N]	

@BEND Strukturblock-Ende

Funktion

@BEND definiert das Block-Ende für alle Typen von Strukturblöcken.

Format

Name	Operation	Operanden
[name]	@BEND	

>>>>> siehe auch @BEGI, @CASE, @CAS2, @CYCL, @IF, @THRU, @WHIL

@BREA Abbrechen einer Schleife

Funktion

@BREA definiert einen Ausgang aus einer Schleife mit freier Endebedingung oder einer Zählschleife mit freier Endebedingung.

Format

Name	Operation	Operanden
[name]	@BREA[K]	

Beschreibung

@BREA tritt in Verbindung mit dem Makroaufruf @WHEN auf. Trifft die über @WHEN definierte Bedingung zu, wird der Strukturblock mit @BREA verlassen. Trifft die Bedingung nicht zu, wird die auf @BREA folgende Instruktion ausgeführt.

Beispiel

```
Name        Operation   Operanden
LOOP        @CYCL
            .
            .
            @WHEN       NE
            CLI         OK,C'Y'
            @BREA                          (01)
            .
            .
            @WHEN       LZ
            SR          R7,R5
            @BREA                          (02)
            .
            .
            @BEND
```

Das Beispiel zeigt eine Schleife mit zwei Endebedingungen und entsprechend den zwei möglichen Ausgängen (01) und (02). Der Strukturblock wird beim ersten Zutreffen einer der beiden Bedingungen verlassen.

>>>>> siehe auch @CYCL und @WHEN

@CASE Fallunterscheidung durch Nummer

Funktion

@CASE bildet den Kopf einer Mehrfachverzweigung. Der auszuführende Unterblock wird durch die Angabe seiner Nummer angesteuert.

Format

Name	Operation	Operanden
[name]	@CASE	(reg)

name Name

reg Mehrzweckregister, das einen der folgenden Werte als positiven absoluten Ausdruck enthält:

- dezimaler selbstdefinierender Wert
- vordefinierter Name eines Registers
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde.

Beschreibung

Der Inhalt von reg gibt an, der wievielte Unterblock in einer Fallunterscheidung ausgeführt werden soll. Es sind maximal 90 Unterblöcke möglich. Der angegebene Unterblock wird direkt angesprungen.

Die Verwendung von Register 0 ist unzulässig.

Programmierhinweise

1. reg muß vor dem Aufruf von @CASE mit der Nummer des entsprechenden Unterblocks geladen werden.
2. Bei der Verwendung eines Mehrzweckregisters für reg sind die Registerkonventionen zu beachten (siehe A3.3).
3. Ist der Inhalt des @CASE-Registers kleiner als 1 oder größer als die Zahl der definierten Unterblöcke,
   - wird der letzte Unterblock ausgeführt, falls im Prozedurkopf CHECK=ON gesetzt ist (siehe 10.2, @ENTR);
   - treten Programmfehler auf, falls im Prozedurkopf CHECK=OFF gesetzt ist.

Beispiel

```
Name      | Operation | Operanden
----------+-----------+-----------------------------------
          | L         | R6,T2
          | @CASE     | (R6)
          | @BEGI     |                  ┐
          | .         |                  } 1. Unterblock
          | .         |                  ┘
          | @BEND     |
          | @BEGI     |                  ┐
          | .         |                  } 2. Unterblock
          | .         |                  ┘
          | @BEND     |
          | .         |                  ┐ weitere Unterblöcke
          | .         |                  } möglich
          | @BEND     |
          | .         |
          | .         |
T1        | DC        | F'1'
T2        | DC        | F'2'
          | .         |
          | .         |
```

Im Beispiel wird Register 6 für die @CASE-Verzweigung verwendet. Der Inhalt von Register 6 ist hier 2, deshalb wird der zweite Unterblock ausgeführt.

>>>>> siehe auch @BEGI und @BEND

@CAS2 Fallunterscheidung durch Vergleich

Funktion

@CAS2 bildet den Kopf einer Mehrfachverzweigung. Der auszuführende Unterblock wird durch die Angabe eines Selektors angesteuert.

Format

Name	Operation	Operanden
[name1]	@CAS2	{name2 | literal | (reg)}[,COMP=instr]

name1 Name
name2 Name des Feldes, das den Selektor enthält
literal Literal, das den Selektor direkt angibt (siehe 2.5.3)
reg Mehrzweckregister, das den Selektor in Form eines der folgenden Werte enthält:
- dezimaler selbstdefinierender Wert
- vordefinierter Name eines Registers
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

instr Assemblerbefehl, der eine Anzeige setzt (siehe Assemblerbefehle [3]).

Beschreibung

Der Selektor im @CAS2-Makro wird mit den Komparanden der @OF-Makros verglichen. Bei Gleichheit von Selektor und Komparand wird der entsprechende Unterblock ausgeführt. Die Anzahl der Unterblöcke ist nicht begrenzt.

Mit COMP=instr kann eine vom Standard abweichende Vergleichsart gewählt werden.

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-----------------------------
            | @CAS2     | TEST
            | @OF       | =X'0005'
            | .         |                          } 1.
            | .         |                          } Unterblock
            | @OF       | =X'0006'
            | .         |                          } 2.
            | .         |                          } Unterblock
            | @OFRE     |
            | .         |                          } Rest-
            | .         |                          } unterblock
            | @BEND     |
```

Das Feld TEST enthält den Selektor. Ist der Selektor gleich dem ersten Komparanden (X'0005'), wird der erste Unterblock ausgeführt. Ist der Selektor gleich dem zweiten Komparanden (X'0006'), wird der zweite Unterblock ausgeführt. In allen anderen Fällen wird der Restunterblock verarbeitet.

>>>>> siehe auch @OF, @OFRE und @BEND

@CONDI Contingency-Routine abmelden

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@CONDI (**Con**tingency **di**sable) meldet eine Contingency-Routine ab (siehe auch Abschnitt 9.6.4, Folgeprozesse behandeln).

Format

Name	Operation	Operanden
[name]	@CONDI	CONID={symb Adr / (reg)}

name Name
symb Adr symbolische Adresse eines Feldes
reg Mehrzweckregister, enthält die Adresse der Kurzkennung.

Beschreibung

CONID= Kurzkennung der Routine

symb Adr
: symbolische Adresse eines Feldes der Länge 4 auf Wortgrenze mit der Kurzkennung. Die Kurzkennung wird vom Makro @CONEN geliefert.

(reg) wahlweise Register mit der Adresse der Kurzkennung.

Der Inlinecode des Makros übernimmt die Angaben aus dem Aufruf in einen ILCS-konformen Parameterblock; mit dieser Versorgung wird der entsprechende Entry im Standard-Contingency-Handler zum Abmelden der Contingency-Routine angesprungen.

Nach Rückkehr des Standard-Contingency-Handlers in die aufrufende Prozedur enthält R15 einen Returncode, der besagt, ob die Funktion ausgeführt wurde oder nicht bzw. welche Fehler auftraten.

@CONEN Contingency-Routine anmelden

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@CONEN (**Con**tingency **en**able) meldet eine Contingency-Routine an (siehe auch 9.6.4, Folgeprozesse behandeln).

Format

Name	Operation	Operanden
[name]	@CONEN	CONAME={name1 | symb Adr1 | (reg1)} [CONLEN=länge] CONID={symb Adr2 | (reg2)} CONADR={symb Adr3 | (reg3)} CONMSG={symb Adr4 | (reg4)} [CONLEV={level | (reg5)}]

name Name
name1 Name der Contingency-Routine

symb Adr1...symb Adr4
symbolische Adressen von Feldern

reg1...reg4
Mehrzweckregister, in denen die Adresse eines Feldes steht.

länge Länge in Bytes
level selbstdefinierender dezimaler Wert
reg5 Mehrzweckregister, das eine Prioritätsangabe enthält.

Beschreibung

CONAME= Name der Contingency-Routine

name1 Der Name der Contingency-Routine besteht aus einem String mit max. 54 Zeichen. Blank beendet den Namen.

symb Adr1
Der Name der Contingency-Routine ist in einem Feld enthalten, dessen symbolische Adresse angegeben wird.

(reg1) Die Adresse des Namens für die Contingency-Routine ist in einem Register enthalten.

CONLEN= Länge des Namens der Routine

länge Länge des Namens in Bytes; nur nötig bei Übergabe einer Adresse für den Namen der Contingency-Routine.
Voreinstellung bei CONAME=symb Adr1: Längenattribut der symbolischen Adresse
Voreinstellung bei CONAME=(reg1): 54 Bytes

CONID= Zurückgemeldete Kurzkennung

symb Adr2
Gibt die symbolische Adresse eines Feldes der Länge 4 auf Wortgrenze an, in dem die Kurzkennung zurückgemeldet werden soll.

(reg2) Gibt ein Register an, das die Adresse eines Feldes der Länge 4 auf Wortgrenze enthält.

CONADR= Startadresse der Contingency-Routine

symb Adr3
Die Startadresse der Contingency-Routine ist in einem Feld auf Wortgrenze enthalten, dessen symbolische Adresse angegeben wird.

(reg3) Die Startadresse der Contingency-Routine ist in einem Register enthalten.

CONMSG= Mitteilung des Benutzers

symb Adr4
Die Mitteilung des Benutzers an die Contingency-Routine ist in einem Feld der Länge 4 auf Wortgrenze enthalten, dessen symbolische Adresse angegeben wird.

(reg4) Die Adresse der Mitteilung des Benutzers an die Contingency-Routine ist in einem Register enthalten.

CONLEV= Priorität der Contingency-Routine

level Die Priorität der Contingency-Routine wird als selbstdefinierender dezimaler Wert von 1 - 126 angegeben.
Voreinstellung ist 1.

(reg5) Ein Register enthält die Prioritätsangabe der Contingency-Routine.

Der Inlinecode des Makros übernimmt die Angaben aus dem Aufruf in einen ILCS-konformen Parameterblock; mit dieser Versorgung wird der entsprechende Entry im Standard-Contingency-Handler zum Anmelden der Contingency-Routine angesprungen.

Nach Rückkehr des Standard-Contingency-Handlers in die aufrufende Prozedur enthält R15 einen Returncode, der besagt, ob die Funktion ausgeführt wurde oder nicht bzw. welche Fehler auftraten.

@CYCL Schleifen-Kopf

Funktion

@CYCL bildet den Kopf einer Schleifenkonstruktion. Die Anzahl der Durchläufe wird mit einem Wiederholungsfaktor oder durch eine Endebedingung bestimmt.

Format 1: Schleife mit freier Endebedingung

Name	Operation	Operanden
[name]	@CYCL[E]	

>>>>> siehe auch @BREA, @WHEN und @BEND

Format 2: Zählschleife und Zählschleife mit freier Endebedingung

Name	Operation	Operanden
[name]	@CYCL[E]	(reg)

name Name
reg Mehrzweckregister; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

Beschreibung

Der Inhalt von reg gibt die Zahl der Wiederholungen des Schleifenunterblocks an. Der Maximalwert für die Wiederholungszahl ist X'FFFFFFFF'.

reg wird nach jedem Durchlauf um 1 vermindert und auf 0 abgefragt. Ist 0 erreicht, wird der Strukturblock verlassen.

Programmierhinweise

1. Bei der Angabe von reg müssen die Registerkonventionen beachtet werden (siehe 11.3).
2. reg darf im Schleifenunterblock nicht verändert werden.

Beispiel

Das Beispiel zeigt eine Zählschleife mit freier Endebedingung. Die Schleife wird in diesem Fall zehnmal durchlaufen, wenn nicht vorher die Abbruchbedingung eintritt.

```
Name     | Operation | Operanden
---------+-----------+-----------------
         | L         | R7,NR              (01)
LOOP     | @CYCL     | (R7)               (02)
         | .         |
         | .         |
         | @WHEN     | EQ                 ]
         | CLC       | TEST,END           } (03)
         | @BREA     |                    ]
         | .         |
         | .         |
         | @BEND     |
         | .         |
         | .         |
NR       | DC        | F'10'
TEST     | .         | .
END      | .         | .
```

(01) Register 7 wird mit dem Wiederholungsfaktor geladen
(02) Schleifenanfang
(03) Abbruchbedingung und Schleifenausgang

>>>>> siehe auch @BREA, @WHEN und @BEND

@DATA Datenzugriff und Speicheranforderung

Funktion

@DATA realisiert den Zugriff auf Benutzerdaten und die Anforderung des dafür benötigten Speicherplatzes. Abhängig von der Angabe der Speicherklasse werden durch den @DATA-Makro entsprechende Instruktionen generiert.

Format 1: Datenbereiche der Klassen A und C

Name	Operation	Operanden
[name]	@DATA	CLASS={A|C},BASE=reg1 [,DSECT=dsect_name] {,LENGTH={val|(reg2)}} [{,INIT={int_name|(reg3)}|,EXTINIT=ext_name}]

name Name

reg1, reg2, reg3
Mehrzweckregister; positive absolute Ausdrücke, entweder
- dezimale selbstdefinierende Werte oder
- vordefinierte Namen von Registern oder
- Namen, denen ein entsprechender selbstdefinierender Wert zugewiesen wurde

dsect_name
Name eines Pseudoabschnitts, maximal 7 Zeichen

val dezimaler selbstdefinierender Wert

int_name, ext_name
Namen von Datenbereichen

Beschreibung

CLASS=A

Es handelt sich um einen Datenbereich der Klasse Automatic.
CLASS=A ist unzulässig in Prozeduren vom Typ B, L oder D, sowie in allen Prozeduren, bei denen im @ENTR "ENV=C" gesetzt ist.

CLASS=C

Es handelt sich um einen Datenbereich der Klasse Controlled
CLASS=C ist unzulässig in Prozeduren vom Typ B, L oder D.

BASE=reg1

Gibt das zu verwendende Basisadreßregister an.

Beim Aufruf von @DATA wird das genannte Register mit der Adresse des zur Verfügung gestellten Speicherbereichs geladen und als Basisadreß-register bereitgestellt.

DSECT=dsect_name

Gibt den Namen des Pseudoabschnitts an, mit dem der angeforderte Speicherbereich überlagert werden soll.

LENGTH=

Gibt die Länge des anzufordernden Bereichs in bytes durch val direkt an, bzw. durch den Inhalt von reg2.

INIT= Gibt den Namen (int_name) oder die Adresse (in reg3) eines Bereichs im gleichen Modul (intern) an, dessen Daten zur Initialisierung in den angeforderten Speicherbereich kopiert werden sollen.

EXTINIT=ext_name

Gibt den Namen eines Bereichs in einem beliebigen Modul (extern) an, dessen Daten zur Initialisierung in den angeforderten Speicherbereich kopiert werden sollen.

Programmierhinweis

Der in DSECT=dsect_name angegebene Pseudoabschnitt muß mit der folgenden Assembleranweisung abgeschlossen werden:

```
Ldsect_name     EQU   *-dsect_name
```

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-------------------------------------------
EX1         | @ENTR     | TYP=E
            | @DATA     | CLASS=A,BASE=R7,DSECT=DATA,INIT=INDATA
            | .         |
            | .         |
            | @EXIT     |
            | .         |
            | .         |
INDATA      | DS        | ..                  (01)
            | .         |
            | .         |
            | @END      |
DATA        | DSECT     |                     (02)
..          | DC        | ..
            | .         |
            | .         |
LDATA       | EQU       | *-DATA              (03)
            | .         |
            | .         |
```

(01) Definition der Daten, die in den angeforderten Speicherbereich kopiert werden sollen.

(02) Definition des Pseudoabschnitts, der den angeforderten Speicherbereich überlagern soll.

(03) Abschluß des Pseudoabschnitts, Längenberechnung.

Format 2: Datenbereiche der Klasse S

Name	Operation	Operanden
[name]	@DATA	CLASS=S ,BASE=reg [,INIT=int_name] {,EXTINIT=ext_name,DSECT=dsect_name}

name Name

reg Mehrzweckregister; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Werte oder
- vordefinierter Name eines Registers
- Name dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

int_name, ext_name
Namen von Datenbereichen

dsect_name
Name eines Pseudoabschnitts

Beschreibung

CLASS=S
Es handelt sich um einen Datenbereich der Klasse Static.

BASE=reg
Gibt das zu verwendende Basisadreßregister an.

Beim Aufruf von @DATA wird das genannte Register als Basisadreßregister bereitgestellt und mit der Anfangsadresse des Datenbereichs geladen.

INIT=int_name
Gibt den Namen eines Datenbereichs an, der im gleichen Modul liegt (intern).

EXTINIT=ext_name
Gibt den Namen eines Datenbereichs an, der in einem beliebigen Modul liegt (extern).

DSECT=dsect_name
Gibt den Namen des Pseudoabschnitts an, der die Struktur des Bereichs ext_name beschreiben muß.

Beispiel

Das Beispiel zeigt einen Zugriff auf Daten, die in einem externen Modul definiert sind.

```
Name        | Operation | Operanden
------------------------------------------------------------------
* Modul A                                              (01)
            |    .      |
            |    .      |
            | @DATA     | CLASS=S,BASE=R5,EXTINIT=BDATEN,DSECT=ADATEN
            |    .      |
            |    .      |
            | @EXIT     |
            | @END      |
ADATEN      | DSECT     |                               (02)
A1          | DS        | CL5
            |    .      |
            |    .      |
AEND        | DS        | CL6
LADATEN     | EQU       | *-ADATEN
* Modul B                                              (03)
DATEN       | START     |
            | ENTRY     | BDATEN
BDATEN      | DS        | 0CL100                        (04)
B1          | DC        | C'HALLO'
            |    .      |
            |    .      |
BEND        | DC        | C'END  B'
```

(01) Modul A enthält die Prozedur, die auf die Daten zugreifen soll.
(02) Pseudoabschnitt, der die Struktur des externen Datenbereichs beschreibt.
(03) Modul B ist der externe Modul, der die Datendefinitionen enthält.
(04) Datendefinition im externen Modul.

Format 3: Datenbereiche der Klasse B

Name	Operation	Operanden
[name]	@DATA	CLASS=B ,BASE=reg ,DSECT=dsect_name

name Name

reg Mehrzweckregister, positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

dsect_name
Name eines Pseudoabschnitts oder eines Datenbereichs

Beschreibung

CLASS=B
Es handelt sich um einen Datenbereich der Klasse Based.

BASE=reg
Gibt das zu verwendende Basisadreßregister an.

reg muß mit der Anfangsadresse des Datenbereichs geladen sein, für den in einer anderen Prozedur bereits Speicher reserviert wurde.

DSECT=dsect_name
Gibt den Namen eines bereits bestehenden Datenbereichs an oder den Namen eines Pseudoabschnitts, der die Struktur des neuen Datenbereichs beschreibt.

Programmierhinweis

reg muß in einer zeitlich vorangehenden oder dynamisch übergeordneten Prozedur durch einen @DATA-Aufruf mit CLASS=A, C oder S mit der Anfangsadresse des Datenbereiches geladen werden.

Beispiel

```
Name        | Operation | Operanden
------------------------------------------------------------
* übergeordnete Prozedur
FIRST       | @ENTR     | TYP=M
            | @DATA     | CLASS=S,BASE=R9,INIT=CONST     (01)
            | .         |
            | .         |
            | @PASS     | NAME=SECOND                    (02)
            | @EXIT     |
            | @END      |
CONST       | DS        | 0D                             (03)
            | DC        | ..
            | .         |
            | .         |

* untergeordnete Prozedur
SECOND      | @ENTR     | TYP=I
            | @DATA     | CLASS=B,BASE=R9,DSECT=CONST    (04)
            | .         |
            | .         |
```

(01) Einrichten des Datenbereichs CONST mit Basisadressregister R9.
(02) Aufruf der Prozedur SECOND.
(03) Definition des Datenbereichs CONST.
(04) Überlagerung des neuen Datenbereichs mit der Struktur des Datenbereichs CONST

@DO Schleifen-Unterblock

Funktion

@DO kennzeichnet den Beginn des Schleifenunterblocks in einer iterativen Schleife und in einer Schleife mit Vorabprüfung.

Format

Name	Operation	Operanden
[name]	@DO	

>>>>> siehe auch @BEND, @THRU und @WHIL

@ELSE Nein-Unterblock

Funktion

@ELSE bildet den Kopf des Nein-Unterblocks in einer Entscheidung.

Format

Name	Operation	Operanden
[name]	@ELSE	

Beschreibung

Der Unterblock in einer @IF-Verzweigung, der mit @ELSE beginnt, wird nur durchlaufen, wenn die mit @IF gesetzte Bedingung **nicht** zutrifft.

>>>>> siehe auch @IF, @THEN und @BEND

@END Statisches Prozedurende

Funktion

@END kennzeichnet das statische Ende einer durch @ENTR eröffneten Prozedur.

Format

Name	Operation	Operanden
[name]	@END	[,LTORG={YES / NO}][,DROP={(reg[,...]) / ()}]

name Name

reg Mehrzweckregister; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

Beschreibung

Durch den Aufruf des @END-Makro
- wird bei allen Prozeduren, außer Typ D, eine LTORG-Anweisung generiert. D.h., ein Literalbereich wird ab der nächsten Doppelwortgrenze angelegt (siehe 4.2, LTORG-Anweisung).
- werden alle durch @ENTR und @DATA zugewiesenen Basisadreßregister freigegeben (siehe 4.2, DROP-Anweisung).

LTORG=YES
: Bewirkt, daß eine LTORG-Anweisung generiert wird; Standardwert für alle Prozeduren außer Typ D.

LTORG=NO
: Es wird keine LTORG-Anweisung generiert; Standardwert für Prozeduren vom Typ D.

DROP=(reg[,...])
: Zusätzlich zu den standardmäßig freigegebenen Registern können weitere freigegeben werden.

DROP=() Es wird eine DROP-Anweisung ohne Operanden generiert; alle bis dahin mit USING definierten Basisregister werden freigegeben.

@ENTR Prozeduranfang

Funktion

@ENTR bildet den Kopf für alle Prozedurtypen. Abhängig von der Typ-Angabe sind unterschiedliche Operandenangaben möglich.

Für alle folgenden Formate von @ENTR sind zusätzlich zu den angegebenen Operanden die folgenden allgemeinen Angaben möglich.

Format

Name	Operation	Operanden
...	@ENTR	... [,VERS=xxx] [,AUTHOR=name] [,FUNCT='function'] [,CHECK={ON / <u>OFF</u>}] [,TITLE={YES / NO}]

xxx Versionsbezeichnung, wird formatfrei angegeben.

name Name des Programmierers.

function Dokumentationstext; sollte die Funktion der Prozedur angeben; maximale Länge 63 Zeichen.

Beschreibung

CHECK= Entscheidet über die Erzeugung von Prüfungen zur Ablaufzeit mit Fehlerbehandlung.

ON Prüfungen und Fehlerbehandlung werden erzeugt. Die Funktion dient der Programmsicherheit und geht auf Kosten von Speicherplatz und Laufzeit.

OFF Prüfungen und Fehlerbehandlungen werden nicht erzeugt.

TITLE= Steuert die Generierung einer TITLE-Anweisung.

YES TITLE-Anweisung wird erzeugt (Standardwert für alle Prozedurtypen außer D). Der Titel enthält den Namen der Prozedur, ihren Typ, die Version und das Datum der Übersetzung.

NO TITLE-Anweisung wird nicht erzeugt (Standardwert für Prozedurtyp D).

Format 1: Hauptprozedur

Name	Operation	Operanden
name	@ENTR	TYP=M [,MAXPRM=val] [,LOCAL=dsect_name] [,AMODE={24 | 31 | ANY},RMODE={24 | ANY}] [,ENV=C[,LOADR12={YES | NO}]] [,STACK=n] [,ILCS={YES | NO}] [,STREQ=symb Adr1] [,STREL=symb Adr2] [,HPREQ=symb Adr3] [,HPREL=symb Adr4] [,SLTERM=symb Adr5] [,SCTERM=symb Adr6] [,EXTMIN=val1] [,ABKR={YES | NO}] [,PROCHK=symb Adr1] [,ERROR=symb Adr2] [,OTHEVT=symb Adr3]

name Name der Hauptprozedur

val dezimaler selbstdefinierender Wert; Anzahl der zu übergebenden Parameter

dsect_name
Name eines Datenbereichs

symb Adr1...symb Adr6
symbolische Adressen der benutzereigenen Routinen

val1 dezimaler selbstdefinierender Wert, der die minimale Stack-Extent-Größe in Bytes angibt.

symb Adr1...symb Adr3
symbolische Adressen von Behandlungsroutinen

Beschreibung

TYP=M Gibt an, daß es sich um die Hauptprozedur eines Programms handelt.

Werden mehrere Module zu einem ablauffähigen Programm zusammengebunden, darf nur ein Modul eine Prozedur vom TYP=M enthalten.

MAXPRM=val
Muß angegeben werden bei dynamischer Parameterübergabe über die STANDARD-Schnittstelle. val gibt die Maximalzahl der Parameter an, die mit dem @PASS-Makro an die aufgerufene Prozedur übergeben werden sollen (siehe @PASS, Format 3).

Der Speicherplatz für die Parameter wird im Prozedur-STACK reserviert. Die Angabe von MAXPRM= wird ignoriert, wenn gleichzeitig der Operand LOCAL= gesetzt ist, da sich der LOCAL-Bereich und der Bereich für die Parameter im Prozedur-STACK überdecken.

LOCAL=dsect_name
Muß angegeben werden, wenn ein lokaler Datenbereich im Prozedur-STACK bereitgestellt werden soll.

Als Basisadreßregister für die Adressierung dieses Bereichs wird Register 13 verwendet.

Die Struktur des angeforderten Bereichs innerhalb des Prozedur-STACK beschreibt ein Pseudoabschnitt, der mit dsect_name @PAR definiert werden muß (siehe @PAR, Format 3).

AMODE=
ordnet der Prozedur einen Adressierungsmodus zu (siehe 4.2, AMODE-Anweisung).

RMODE=
ordnet der Prozedur ein Ladeattribut zu (siehe 4.2, RMODE-Anweisung).

Bei einer unzulässigen Kombination von AMODE und RMODE wird eine MNOTE generiert und für beide Werte AMODE 24 und RMODE 24 eingesetzt.

ENV=C Nur bei ILCS=NO zugelassen!
muß angegeben werden, wenn sich die betreffende Prozedur wie ein C-Programm verhalten soll. D.h., die Prozedur läuft mit der Steuerung des C-Laufzeitsystems (siehe ASSEMBH, Benutzerhandbuch [1]).

LOADR12=YES
Die Adresse des Program-Manager für C-Programme (siehe C-Compiler, Benutzerhandbuch [9]) soll in Register 12 geladen werden.

LOADR12=NO
Ein Laden von Register 12 erfolgt nicht.

Register 12 muß bereits die Adresse des Program-Manager enthalten. Dies ist immer der Fall, wenn die rufende Prozedur ein C-Programm ist oder wenn in einem Assembler-Programm mit "ENV=C" Register 12 nicht verändert wurde.

STACK=n Größe des dynamisch erweiterbaren Initialstacks in Bytes.

Bei fehlender Angabe wird ein Initialstack von einer Seite (4096 Bytes) angefordert.

ILCS=YES Anschluß an ILCS

ILCS=NO Voreingestellter Wert
Nicht-ILCS-Prozedur

ILCS ermöglicht es dem Anwender bei der Initialisierung benutzereigene Routinen für die Speicherbeschaffung und Speicherfreigabe sowie zur Beendigungsbehandlung anzumelden. Diese Routinen müssen ILCS-Konventionen genügen. Außerdem kann die minimale Stack-Extent-Größe angegeben werden. Diese Angaben gelten für die gesamte ILCS-Umgebung und können nur in der Hauptprozedur @ENTR mit TYP = M und ILCS=YES definiert werden, dazu folgende Operanden:

STREQ=

symb Adr1
symbolische Adresse der benutzereigenen Speicherbeschaffungsroutine für die Stackverwaltung

Eingabe-Parameter:	Bytelänge des Speicherbereichs, mit Ausrichtung auf Doppelwortgrenze
Returnwert in R0:	Zeiger auf Speicherbereich
Returncode in R15:	=F'0', kein Fehler aufgetreten

keine Angabe
Speicherbeschaffung in ILCS über REQM

STREL=

symb Adr2
: symbolische Adresse der benutzereigenen Speicherfreigaberoutine für die Stackverwaltung
  Eingabe-Parameter: Zeiger auf Speicherbereich
  Bytelänge des Speicherbereichs
  Returncode in R15: =F'0', kein Fehler aufgetreten

keine Angabe
: Speicherfreigabe in ILCS über RELM

HPREQ=

symb Adr3
: symbolische Adresse der benutzereigenen Speicherbeschaffungsroutine für die Heapverwaltung
  Eingabe-Parameter: Bytelänge des Speicherbereichs, mit Ausrichtung auf Doppelwortgrenze
  Returnwert in R0: Zeiger auf Speicherbereich
  Returncode in R15: =F'0', kein Fehler aufgetreten

keine Angabe
: Speicherbeschaffung in ILCS über REQM

HPREL=

symb Adr4
: symbolische Adresse der benutzereigenen Speicherfreigaberoutine für die Heapverwaltung
  Eingabe-Parameter: Zeiger auf Speicherbereich
  Bytelänge des Speicherbereichs
  Returncode in R15: =F'0', kein Fehler aufgetreten

keine Angabe
: Speicherfreigabe in ILCS über RELM

SLTERM=

symb Adr5
: symbolische Adresse der benutzereigenen Beendigungsroutine

keine Angabe
: keine benutzereigene Beendigungsroutine

SCTERM=

symb Adr6
: symbolische Adresse der benutzereigenen Beendigungsroutine für STXIT- und Contingency-Prozesse
in R15 Fehlercode: >0 interner Fehler
in R0 und R1 Informationscode:
= 0: es handelt sich um einen STXIT-Prozeß
= 1: es handelt sich um einen Contingency-Prozeß

keine Angabe
: bei internen Fehlern Abbruch des STXIT- oder Contingency-Prozeß mit TERM UNIT=STEP

EXTMIN=

val1
: dezimaler selbstdefinierender Wert, der die minimale Stack-Extent-Größe in Bytes angibt. Die Angabe wird auf die nächste Zweierpotenz von Seiten (4096 Bytes) aufgerundet.

keine Angabe
: minimale Stack-Extent-Größe beträgt 16 Seiten.

ILCS stellt mit dem Standard-Event-Handler (SEH) eine Routine zur Verfügung, die es ermöglicht, in ILCS-Programmen auftretende Ereignisse zu koordinieren (siehe auch 9.6.3, Ereignisse behandeln). Unter der Voraussetzung ILCS=YES sind folgende Operanden möglich:

ABKR= Abbruchkennzeichen setzen/nicht setzen

YES
: In der zugehörigen Event-Handler-List (EHL) der Prozedur wird ein Kennzeichen für den Standard-Event-Handler eingetragen, die Suche nach Behandlungsroutinen innerhalb der Save-Area-Verkettung abzubrechen.

NO
: Das Abbruchkennzeichen wird nicht gesetzt.

PROCHK=

symb Adr1
: symbolische Adresse einer Behandlungsroutine für STXIT-Ereignisse der Klasse ’PROCHK’.

keine Angabe
: dem Standard-Event-Handler wird keine Behandlungs-Routine für Klasse ’PROCHK’ bereitgestellt.

ERROR=

symb Adr2
: symbolische Adresse einer Behandlungsroutine für STXIT-Ereignisse der Klasse ’ERROR’.

keine Angabe
: dem Standard-Event-Handler wird keine Behandlungs-Routine für Klasse ’ERROR’ bereitgestellt.

OTHEVT=

symb Adr3
: symbolische Adresse einer Behandlungsroutine für Nicht-STXIT-Ereignisse.

keine Angabe
: dem Standard-Event-Handler wird keine Behandlungs-Routine für Klasse ’OTHEVT’ bereitgestellt.

Programmierhinweise

1. Das Abbruchkennzeichen in der Event-Handler-List wird benötigt, wenn der Standard-Event-Handler die Suche nach Ereignisbehandlungsroutinen abbrechen muß, da innerhalb der Prozedurverschachtelung ein Wechsel von einer ILCS- zu einer Nicht-ILCS-Prozedur stattfindet.
2. Wird mit TEST-SUPPORT=YES übersetzt, so darf vor dem Aufruf @ENTR Typ=M auf keinen Fall eine CSECT-Anweisung mit einem Namen ungleich des Namenseintrages des @ENTR stehen, da die für AID erzeugte Konsistenzkonstante diese CSECT abschließt. Dies kann zu einem undefinierten Programmverhalten führen, wenn der Programmabschnitt nicht vor der Konstanten verlassen wird.

Format 2: Prozeduren vom Typ I oder E

Name	Operation	Operanden
name1	@ENTR	TYP={I / E} [,PLIST=({name2 / (reg)}[,...])] [,PASS={OPT[IMAL] / STA[NDARD]}] [,MAXPRM=val] [,LOCAL=dsect_name] [,RETURNS={YES / NO}] [,ENTRY={ENTRY / CSECT[,AMODE={24 / 31 / ANY},RMODE={24 / ANY}]}] [,ENV=C[,LOADR12={YES / NO}]] [,ILCS={YES / NO}] [,ABKR={YES / NO}] [,PROCHK=symb Adr1] [,ERROR=symb Adr2] [,OTHEVT=symb Adr3]

name1 Name der Prozedur
name2 Formalparameter
reg Register als Formalparameter; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

val dezimaler selbstdefinierender Wert; Anzahl der zu übergebenden Parameter
dsect_name
Name eines Datenbereichs

symb Adr1...symb Adr3
symbolische Adressen von Behandlungsroutinen

Beschreibung

TYP=I Gibt an, daß es sich um eine Prozedur handelt, die nur von dem Modul aus gerufen werden kann, in dem sie selbst liegt (interne Prozedur) und die an Speicherverwaltung und Registersicherung angeschlossen ist.

TYP=E Gibt an, daß es sich um eine Prozedur handelt, die von einem beliebigen Modul aus gerufen werden kann (externe Prozedur) und die an Speicherverwaltung und Registersicherung angeschlossen ist.

PLIST= Wird angegeben für die Übernahme von Parametern.

Der Operand gibt die Formalparametern an, eine Liste von Datenfeldern und Registern, in die beim Aufruf der Prozedur Einträge übernommen werden (siehe @PAR, Format 2).

Bei einer Übernahme der Parameter in den LOCAL-Bereich der Prozedur müssen die PLIST-Operanden mit den PLIST-Operanden des zugehörigen @PAR übereinstimmen.

PASS=OPT
Nur bei ILCS=NO zugelassen!
Muß angegeben werden, wenn in der aufgerufenen Prozedur die Parameterübernahme über die OPTIMAL-Schnittstelle erfolgen soll.

Zur Parameterübernahme werden die Register 1 bis 4 verwendet.

PASS=STA

In der aufgerufenen Prozedur erfolgt die Parameterübernahme über die STANDARD-Schnittstelle.

Zur Parameterübernahme wird nur das Register 1 verwendet. Register 1 enthält die Adresse der Parameterliste.

Die Angabe PASS= muß mit dem PASS-Operanden im @PASS-Makro der aufrufenden Prozedur übereinstimmen (siehe @PASS, Format 3).

MAXPRM=val

Muß angegeben werden bei dynamischer Parameterübergabe über die STANDARD-Schnittstelle. val gibt die Maximalzahl der Parameter an, die mit dem @PASS-Makro an eine Prozedur übergeben werden können (siehe @PASS, Format 3).

LOCAL=dsect_name

Muß angegeben werden, wenn ein lokaler Datenbereich im Prozedur Prozedur-STACK bereitgestellt werden soll.

Basisadreßregister für diesen Bereich ist Register 13.

Die Struktur des angeforderten Bereichs beschreibt ein Pseudoabschnitt, der mit dsect_name @PAR definiert werden muß. dsect_name kann dabei

- einen @PAR kennzeichnen, der einen lokalen Pseudoabschnitt definiert (Format 3 von @PAR) oder
- einen @PAR kennzeichnen, der einen Pseudoabschnitt zur Parameterübernahme definiert (Format 2 von @PAR).

RETURNS=

Regelt, welche Register beim Verlassen der Prozedur über @EXIT wieder mit den ursprünglichen Werten (vor Aufruf der Prozedur) geladen werden bzw. ob ein Funktionswert zurückgeliefert wird.

Der Operand ILCS=YES/NO beeinflußt die Funktionsweise von RETURNS=YES/NO.

YES Bei ILCS=NO:
Die Register 0 und 2 bis 14 werden zurückgeladen, Register 1 nicht. Somit kann Register 1 dazu verwendet werden, der rufenden Prozedur einen Funktionswert zu übermitteln. Die gerufene Prozedur wird dadurch zu einer "Funktionsprozedur".

Bei ILCS=YES:
Die Prozedur liefert einen Funktionswert zurück. Die Register 0 bis 14 werden zurückgeladen. Der Funktionswert im Register 1 wird im Prozedurepilog nach Register 0 kopiert, wenn der Rufer eine ILCS-Prozedur ist.

NO Bei ILCS=NO:
Alle Register 0 bis 14 werden zurückgeladen.

bei ILCS=YES:
Die Prozedur liefert keinen Funktionswert zurück und damit wird im Prozedurepilog R1 auch nicht nach R0 kopiert. Die Registerstände von R0 und R1 sind undefiniert. Register 2 bis 14 werden zurückgeladen.

Keine Angabe
Die Register 2 bis 14 werden zurückgeladen.

Das Rückladen der Register kann bei ILCS=NO zusätzlich mit dem Operanden RESTORE=MIN des @EXIT-Makro eingeschränkt werden.

ENTRY= Gibt an, welche Eingangs-Instruktionen für die Prozedur generiert werden sollen.

Die Angabe von ENTRY= ist nur bei Prozeduren vom Typ E zulässig.

bei Angabe von ENTRY wird generiert:

```
name1   DS      0D
        ENTRY   name1
```

Diese Angabe ist notwendig, wenn für das Testen eines Programms mit mehr als einer Prozedur die AID-Kommandos %CONTROL oder %TRACE verwendet werden sollen (siehe AID, Testen von ASSEMBH-Programmen [2]).

Bei Angabe von CSECT wird generiert: `name1 CSECT`

AMODE= ordnet der Prozedur einen Adressierungsmodus zu
(siehe 4.2, AMODE-Anweisung).

RMODE= ordnet der Prozedur ein Ladeattribut zu (siehe 4.2, RMODE-Anweisung).

Bei einer unzulässigen Kombination von AMODE und RMODE wird eine MNOTE generiert und für beide Werte AMODE 24 und RMODE 24 eingesetzt.

ENV=C muß angegeben werden, wenn sich die betreffende Prozedur wie ein C-Programm verhalten soll; d.h., die Prozedur läuft mit der Steuerung des C-Laufzeitsystems (siehe ASSEMBH, Benutzerhandbuch [1]).

Die Angabe ist nur für Prozeduren vom Typ E mit ILCS=NO zulässig.

LOADR12=YES
Die Adresse des Program-Manager für C-Programme (siehe C-Compiler, Benutzerhandbuch [9]) soll in Register 12 geladen werden.

LOADR12=NO

Ein Laden von Register 12 erfolgt nicht.

Register 12 muß bereits die Adresse des Program-Manager enthalten. Dies ist immer der Fall, wenn die rufende Prozedur ein C-Programm ist oder wenn in einem Assembler-Programm mit "ENV=C" Register 12 nicht verändert wurde.

ILCS=YES Anschluß an ILCS

ILCS=NO Voreingestellter Wert
Nicht-ILCS-Prozedur

ILCS stellt mit dem Standard-Event-Handler (SEH) eine Routine zur Verfügung, die es ermöglicht, in ILCS-Programmen auftretende Ereignisse zu koordinieren (siehe auch 9.6.3, Ereignisse behandeln).
Falls ILCS=YES gesetzt ist, sind folgende Angaben möglich:

ABKR= Abbruchkennzeichen setzen/nicht setzen

YES In der zugehörigen Event-Handler-List der Prozedur wird ein Kennzeichen für den Standard-Event-Handler eingetragen, die Suche nach Behandlungsroutinen innerhalb der Save-Area-Verkettung abzubrechen.

NO Das Abbruchkennzeichen wird nicht gesetzt.

PROCHK=

symb Adr1

symbolische Adresse einer Behandlungsroutine für STXIT-Ereignisse der Klasse ’PROCHK’.

keine Angabe

dem Standard-Event-Handler wird keine Behandlungs-Routine für Klasse ’PROCHK’ bereitgestellt.

ERROR=

symb Adr2
: symbolische Adresse einer Behandlungsroutine für STXIT-Ereignisse der Klasse ’ERROR’.

keine Angabe
: dem Standard-Event-Handler wird keine Behandlungs-Routine für Klasse ’ERROR’ bereitgestellt.

OTHEVT=

symb Adr3
: symbolische Adresse einer Behandlungsroutine für Nicht-STXIT-Ereignisse.

keine Angabe
: dem Standard-Event-Handler wird keine Behandlungs-Routine für Klasse ’OTHEVT’ bereitgestellt.

Programmierhinweis

Das Abbruchkennzeichen in der Event-Handler-List wird benötigt, wenn der Standard-Event-Handler die Suche nach Ereignisbehandlungsroutinen abbrechen muss, da innerhalb der Prozedurverschachtelung ein Wechsel von einer ILCS- zu einer Nicht-ILCS-Prozedur stattfindet.

Beispiel für Format 1 und Format 2

```
Name        | Operation | Operanden
------------+-----------+-------------------------------------------
* rufende Prozedur
MAIN        | @ENTR     | TYP=M,MAXPRM=2,LOCAL=DUMMY                  (01)
            | .         |
            | .         |
            | @PASS     | NAME=PRO2,PLIST=(FIELD1,FIELD2)             (02)
            | .         |
            | .         |
DUMMY       | @PAR      | D=YES                                       ┐
NAME1       | DS        | ..                                          │
            | .         |                                             } (03)
            | .         |                                             │
DUMMY       | @PAR      | LEND=YES                                    ┘
            | .         |
            | .         |
            |           |
* gerufene Prozedur
PRO2        | @ENTR     | TYP=I,LOCAL=IN,PLIST=(INFIELD1,INFIELD2)    (04)
            | .         |
            | .         |
IN          | @PAR      | D=YES,LEND=YES,PLIST=(INFIELD1,INFIELD2)    (05)
            | .         |
            | .         |
```

(01) Die Prozedur MAIN
- soll maximal 2 Parameter an eine andere übergeben (MAXPRM=2) und
- für sie wird ein lokaler Datenbereich angefordert, dessen Struktur im Pseudoabschnitt DUMMY beschrieben wird.

(02) Aufruf von PRO2, es sollen 2 Parameter an PRO2 übergeben werden; die Übergabe erfolgt dynamisch über die STANDARD-Schnittstelle.

(03) Definition des Pseudoabschnitts DUMMY.

(04) PRO2 übernimmt 2 Parameter; der Pseudoabschnitt IN beschreibt die Struktur des Datenbereichs, der für die Übernahme bereit gestellt wird, PLIST=... gibt die Liste der Parameter an.

(05) Definition des Pseudoabschnitts IN.

Format 3: Prozeduren vom Typ B oder L

Name	Operation	Operanden
name	@ENTR	TYP={B / L} [,BASE=reg] [,LOADSB={YES / NO}] [,ENTRY={ENTRY / CSECT[,AMODE={24 / 31 / ANY},RMODE={24 / ANY}]}]

name Name der Prozedur

reg Mehrzweckregister; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

Beschreibung

TYP=B Gibt an, daß es sich um eine Prozedur handelt, die von einem beliebigen Modul aus gerufen werden kann (externe Prozedur) und die nicht Speicherverwaltung und Registersicherung angeschlossen ist.

TYP=L Gibt an, daß es sich um eine Prozedur handelt, die nur vom gleichen Modul aus gerufen werden kann (interne Prozedur) und die nicht an Speicherverwaltung und Registersicherung angeschlossen ist.

BASE=reg Weist der Prozedur ein Register als Basisadreßregister zu.

Ohne diese Angabe wird der Prozedur vom Assembler das Register 15 als Basisadreßregister zugewiesen.

LOADSB=

Regelt das Laden des Basisadreßregisters nach jedem Aufruf einer weiteren Unterprozedur mit @PASS.

YES Das mit dem BASE-Operanden angegebene Basisadreßregister oder das Standard-Basisadreßregister 15 wird mit der Adresse der rufenden Prozedur geladen.

NO Kein Register wird nach dem Aufruf geladen, auch nicht Register 15.

Keine Angabe
Register 15 wird nach dem Aufruf mit der Adresse der rufenden Prozedur geladen.

ENTRY=
gibt an, welche Eingangs-Instruktionen für die Prozedur generiert werden sollen.

Die Angabe von ENTRY= ist nur bei Prozeduren vom Typ B zulässig.

bei Angabe von ENTRY wird generiert:

```
name1  DS       0D
       ENTRY    name1
```

bei Angabe von CSECT wird generiert:

```
name1  CSECT
```

AMODE=
ordnet der Prozedur einen Adressierungsmodus zu (siehe 4.2, AMODE-AMODE-Anweisung).

RMODE=
ordnet der Prozedur ein Ladeattribut zu (siehe 4.2, RMODE-Anweisung). sung).

Bei einer unzulässigen Kombination von AMODE und RMODE wird eine MNOTE generiert und für beide Werte AMODE 24 und RMODE 24 eingesetzt.

Programmierhinweise

1. Ein Basisadreßregister, das mit BASE=reg zugewiesen wurde, muß am Anfang der Prozedur (nach dem @ENTR-Makro) explizit geladen werden, z.B. mit dem Assemblerbefehl: `LR reg,R15`
2. Der Prozedurtyp ist nur bei ILCS=NO zugelassen.

Format 4: Prozeduren vom Typ D

Name	Operation	Operanden
[name]	@ENTR	TYP=D

name Name der Prozedur

Beschreibung

Es handelt sich um eine Prozedur vom Typ D, ohne Anschluß an Speicherverwaltung und Registersicherung. Die Prozedur kann extern oder intern sein.

Programmierhinweise

1. Ist die Prozedur die erste eines Moduls, darf kein Prozedurname angegeben werden. Die Prozedur erhält den Namen der START- oder CSECT-Anweisung.
2. Der Prozedurtyp ist nur bei ILCS=NO zugelassen.

@EVTLC Layout der Kontext-Beschreibung definieren

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@EVTLC (**Ev**en**t L**ayout **C**ontext) definiert das Layout der Kontext-Beschreibung für den Makro @EVTOE.

Format

Name	Operation	Operanden
[name]	@EVTLC	[pre]=zeichenfolge

name Name
pre Präfix
zeichenfolge
alphanumerische Zeichen

Beschreibung

pre= Präfix für die symbolischen Namen der Einzelfelder

zeichenfolge
max. 4 Zeichen
pre muß der Assemblersyntax für Namen genügen.

Mit dem Präfix werden folgende DS-Anweisungen für die Einzelfelder der Kontextbeschreibung definiert:

```
preCR0     DS    F        Context Register
preCR1     DS    F
preCR2     DS    F
preCR3     DS    F
preCR4     DS    F
preCR5     DS    F
preCR6     DS    F
preCR7     DS    F
preCR8     DS    F
preCR9     DS    F
preCR10    DS    F
preCR11    DS    F
preCR12    DS    F
preCR13    DS    F
preCR14    DS    F
preCR15    DS    F
preCPC     DS    F        Program Counter
preCEVC    DS    F        Event Code
preCFP0    DS    2F       Floating Point Register
preCFP2    DS    2F
preCFP4    DS    2F
preCFP6    DS    2F
preCILC    DS    X        Instruction Length Code
preCCC     DS    X        Condition Code
preCPM     DS    X        Program Mask
preCSS     DS    X        Sprachschlüssel
```

Programmierhinweis

Ein Präfix mit mehr als 4 Zeichen wird auf 4 Zeichen gekürzt.

@EVTOE Nicht-STXIT-Ereignis signalisieren

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@EVTOE (**Ev**en**t**ing **o**ther **e**vent) signalisiert ein Nicht-STXIT-Ereignis (siehe auch 9.6.3, Ereignisse behandeln).

Format

Name	Operation	Operanden
[name]	@EVTOE	CONTXT={symb Adr / (reg)}

name Name
symb Adr symbolische Adresse eines Datenbereiches
reg Mehrzweckregister, enthält die Adresse des Datenbereiches

Beschreibung

CONTXT= beschreibt die Umgebung der Ereignisstelle

symb Adr
: symbolische Adresse eines Datenbereiches der Länge 112 Bytes. In diesen Bereich muß der Benutzer den Kontext der Programmstelle ablegen, an der das Ereignis auftrat. Dazu gehören die Inhalte aller Standard-(Kontext-) und Gleitpunktregister sowie der Befehlszähler, die Programmaske und der vom Benutzer zu definierende Ereigniscode für die von ILCS aufgerufene Behandlungsroutine.

(reg) wahlweise Register, das die Adresse des Datenbereiches enthält.

Zur Struktur dieses Datenbereiches siehe Makro @EVTLC.

Der Inlinecode des Makros übernimmt die Angaben aus dem Aufruf in einen ILCS-konformen Parameterblock; mit dieser Versorgung wird der entsprechende Entry im Standard-Event-Handler zum Signalisieren eines Nicht-STXIT-Ereignisses angesprungen.

Nach Rückkehr des Standard-Event-Handlers in die aufrufende Prozedur enthält R15 einen Returncode, der besagt, ob die Funktion ausgeführt wurde oder nicht bzw. welche Fehler auftraten.

Programmierhinweis

Eine Abprüfung der Kontexteinzelfelder ist nicht möglich; deshalb ist der Benutzer für die Richtigkeit und Konsistenz der Kontextangaben verantwortlich.

@EXIT Dynamisches Prozedurende

Funktion

@EXIT beendet die aufgerufene Prozedur und gibt die Steuerung an die aufrufende Prozedur zurück.

Format 1: Rückkehr aus Prozeduren vom Typ M: B oder L

Name	Operation	Operanden
[name]	@EXIT	[LOADR12={YES / NO}]

Beschreibung

In Prozeduren vom Typ M bewirkt @EXIT die Beendigung des Programms.

In Prozeduren von Typ B oder L wird das Programm mit der Instruktion fortgesetzt, die in der rufenden Prozedur dem @PASS-Makro folgt.

LOADR12=

kann nur angegeben werden für Prozeduren, bei denen "ENV=C" und ILCS=NO gesetzt ist.

YES Die Adresse des Program-Manager für C-Programme (siehe C-Compiler, Benutzerhandbuch [9]) soll in Register 12 geladen werden.

NO Ein Laden von Register 12 erfolgt nicht.

Register 12 muß bereits die Adresse des Program-Manager enthalten. Dies ist immer der Fall, wenn die rufende Prozedur ein C-Programm ist oder wenn in einem Assembler-Programm mit "ENV=C" Register 12 nicht verändert wurde.

Format 2: Rückkehr aus Prozeduren vom Typ E oder I

Name	Operation	Operanden
[name1]	@EXIT	[RC=[name2 / (reg) / val] / TO=proc_name / RESTORE=MIN / PROG=FORTRAN / LOADR12=[YES / NO]][,...]

name1 Name

name2 Name eines Datenfeldes, das den Rückkehrwert enthält

reg Mehrzweckregister, das den Rückkehrwert oder die Adresse des Rückkehrwertes enthält; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

val selbstdefinierender Wert, der den Rückkehrwert direkt angibt

proc_name Name einer Prozedur, die in der Aufrufhierarchie höher liegen muß als die aufrufende Prozedur.

Beschreibung

Nach @EXIT wird das Programm mit der Instruktion fortgesetzt, die in der rufenden Prozedur dem @PASS-Makro folgt.

Durch @EXIT werden die Register 2 bis 14 wieder mit ihren Werten beim Prozeduraufruf geladen.

@EXIT-Aufruf in einer ILCS-Prozedur (@ENTR ILCS=YES)

Falls der @ENTR-Parameter RETURNS=YES gesetzt war, wird der Funktionswert in Register 1 nach Register 0 kopiert. Bei RETURNS=NO unterbleibt das Kopieren des Funktionswertes.

RC= Sollte nur in Prozeduren angegeben werden, bei denen ILCS=NO gesetzt ist.
Gibt einen Rückkehrwert an, der in der gerufenen Prozedur berechnet wurde und an die aufrufende Prozedur übergeben werden soll. Die Übergabe erfolgt standardmäßig im Register 15, bei Nicht-ILCS-FORTRAN-Programmen im Register 0.

Bei Angabe von val wird der Wert, bei Angabe von name2 wird die Adresse übergeben.
Bei Angabe von (reg) wird der Registerinhalt in das Übergaberegister übernommen.

TO=proc_name

Kann nur angegeben werden in Prozeduren, bei denen ILCS=NO gesetzt ist.
Diese Angabe ist erforderlich, wenn nicht in die aufrufende Prozedur zurückgekehrt werden soll, sondern in eine in der Aufrufhierarchie höherliegende.

Das Programm wird mit der Instruktion fortgesetzt, die in der mit proc_name bezeichneten Prozedur dem aktuellen @PASS-Makro folgt.

RESTORE=MIN

Wird angegeben, wenn bei der Rückkehr in die aufrufende Prozedur nur die Register 7 bis 14 zurückgeladen werden sollen.

PROG=FORTRAN

Kann nur angegeben werden in Prozeduren, bei denen ILCS=NO gesetzt ist.
Muß angegeben werden, wenn aus einer gerufenen Assembler-Prozedur in ein rufendes FORTRAN-Programm zurückgekehrt werden soll. In diesem Fall werden die Register 2 bis 14 zurückgeladen.

LOADR12=

kann nur angegeben werden für Prozeduren, bei denen ENV=C und ILCS=NO gesetzt ist.

YES Die Adresse des Program-Manager für C-Programme (siehe C-Compiler, Benutzerhandbuch [9]) soll in Register 12 geladen werden.

NO Ein Laden von Register 12 erfolgt nicht.

Register 12 muß bereits die Adresse des Program-Manager enthalten. Dies ist immer der Fall, wenn die rufende Prozedur ein C-Programm ist oder wenn in einem Assembler-Programm mit "ENV=C" Register 12 nicht verändert wurde.

Programmierhinweis

Der @EXIT Operand RC=<Returncode> bewirkt die Übergabe des Returncodes in Register 15. Innerhalb von strukturierten Assembler-Programmen kann Register 15 wie gewohnt vom Anwender ausgewertet werden (gemäß ILCS-Registerkonventionen müssen im Prozedurepilog lediglich Register 2 bis Register 14 zurückgeladen werden). Rufende Fremdumgebungsprozeduren können diesen Returncode jedoch i.a. nicht mehr abfragen, da gemäß ILCS-Konventionen Register 15 als zerstört angesehen wird. Deshalb sollte auf den @EXIT-Operanden RC möglichst verzichtet werden und der Return-

code sollte anstatt dessen als Funktionswert in Register 1 oder als Ausgabeparameter übergeben werden.

Beispiel

```
Name     | Operation | Operanden
---------+-----------+-------------------------------------
PRO1     | @ENTR     | TYP=M
         | @DATA     | CLASS=S,BASE=R9,INIT=CONST
         | @PASS     | NAME=PRO2                          (01)
         | @IF       | EQ                                 ┐
         | CLI       | TEST,C'A'                          │
         | @THEN     |                                    } (02)
         | .         |                                    │
         | .         |                                    ┘
         | @BEND     |
         | .         |
         | .         |
         | @EXIT     |
         | @END      |
CONST    | DS        | 0H
TEST     | DS        | CL1                                (03)
         | .         |
         | .         |
PRO2     | @ENTR     | TYP=I
         | @DATA     | CLASS=B,BASE=R9,DSECT=CONST
         | .         |                                    ┐
         | .         |                                    } (04)
         | MVI       | TEST,C'A'                          ┘
         | @EXIT     | RC=TEST                            (05)
         | @END      |
```

(01) Aufruf der Prozedur PRO2.
(02) Abhängig vom Rückkehrwert sollen in PRO1 Instruktionen ausgeführt werden.
(03) Definition des Feldes, das den Rückkehrwert enthält.
(04) In PRO2 muß der Rückkehrwert versorgt werden.
(05) Rückkehr in PRO1; übergeben wird die Adresse des Feldes, das den Rückkehrwert enthält.

Die gleiche Funktion könnte erfüllt werden durch die folgenden Instruktionen in PRO2:

```
| LA      | R8,TEST
| @EXIT   | RC=(R8)
```

@FREE Speicherfreigabe

Funktion

@FREE veranlaßt die Feigabe von Speicherplatz, der zuvor mit @DATA CLASS=C angefordert wurde.

Format

Name	Operation	Operanden
[name]	@FREE	BASE=reg

name Name

reg Mehrzweckregister, positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

Beschreibung

reg gibt die Adresse des freizugebenden Speicherbereichs an und muß mit dem BASE-Operanden des zugehörigen @DATA-Makro übereinstimmen.

>>>>> siehe auch @DATA

@IF Entscheidung

Funktion

@IF bildet den Kopf einer Verzweigung, bei der zwischen zwei Alternativen auszuwählen ist.

Format

Name	Operation	Operanden
[name]	@IF	cond_sym

name Name
cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe 9.2.4 und 9.2.5).

Beschreibung

cond_sym legt die Bedingung (siehe 9.2.4 und 9.2.5) fest, die zur Verzweigung verwendet werden soll.

Trifft die Bedingung zu, wird der @THEN-Zweig des Strukturblocks ausgeführt, trifft sie nicht zu, der @ELSE-Zweig.

Beispiel

Name	Operation	Operanden
	@IF	LE
	CR	R1,R2
	@THEN	
	MVI	FIELD,TRUE
	@ELSE	
	MVI	FIELD,FALSE
	@BEND	

Wenn der Inhalt von Register 1 kleiner oder gleich dem von Register 2 ist, wird der @THEN-Unterblock ausgeführt, sonst der @ELSE-Unterblock.

>>>>> siehe auch @THEN, @ELSE und @BEND

@ININ ILCS bei nachgeladenen Moduln aufrufen

Dieser Makroaufruf ist nur innerhalb der ILCS-Umgebung erlaubt. ILCS muß zu einem früheren Zeitpunkt initiiert worden sein.

Funktion

@ININ sorgt durch Aufruf von ILCS dafür, daß bei nachgeladenen (Groß-) Moduln, die ein Laufzeitsystem eingebunden haben, dieses nachträglich initialisiert wird (siehe auch 9.6).

Format

Name	Operation	Operanden
[name]	@ININ	

name Name

Beschreibung

Der Inlinecode des Makros generiert den Ansprung der ILCS-Routine zur Prüfung und ggf. Durchführung der Sprachinitialisierung.

Nach dem Aufruf enthält Register 15 einen Returncode, der besagt, ob die Funktion ausgeführt wurde oder nicht.

Programmierhinweis

Ein Fehler während des Ablaufes der Initialisierungsprüfung, der zu einem undefinierten Programmverhalten führt (z. B. Speicherengpaß), bewirkt den Aufruf der ILCS-Beendigungsroutine. Damit werden auch alle angeschlossenen Sprachen beendet.

@OF Fall-Unterblock

Funktion

@OF bildet in einer Fallunterscheidung durch Vergleich den Kopf eines Unterblocks und gibt den oder die jeweiligen Komparanden an.

Format

Name	Operation	Operanden
[name1]	@OF	{name2 / literal / val}[,...][,COMP=instr]

name1 Name
name2 Name des Feldes, das einen Komparanden enthält
literal Literal, das einen Komparanden direkt angibt (siehe 2.5.3)
val selbstdefinierender Wert, der einen Komparanden direkt angibt
instr Assemblerbefehl, der eine Anzeige setzt (siehe Assemblerbefehle, Beschreibung)

Beschreibung

Der Unterblock, der mit @OF und den jeweiligen Komparanden beginnt, wird ausgeführt, wenn einer der angegebenen Komparanden gleich dem Selektor aus dem @CAS2-Makro ist.

Mit COMP=instr kann für diesen Unterblock eine vom Standard abweichende Vergleichsart gewählt werden.

Programmierhinweise

1. Der Anwender muß sicherstellen, daß Selektor und Komparanden bezüglich Länge, Ausrichtung und Vergleichsbefehl zusammenpassen.
2. Ein selbstdefinierender Wert als Komparand kann nur angegeben werden, wenn mit COMP=instr im @OF-Makro oder im @CAS2-Makro ein Assemblerbefehl angegeben wurde, der als zweiten Operanden einen Direktoperanden erlaubt.

Beispiel

```
Name        | Operation | Operanden
------------+-----------+-------------------
            | @CAS2     | FIELDA              (01)
            | @OF       | FIELDB              (02)
            | .         |
            | .         |
            | @OF       | C'*',COMP=CLI       (03)
            | .         |
            | .         |
```

(01) Beginn der Fallunterscheidung; FIELDA ist der Name des Feldes, das den Selektor enthält.

(02) Beginn des ersten Unterblocks; dieser wird ausgeführt, wenn FIELDA gleich FIELDB ist.

(03) Beginn des zweiten Unterblocks; dieser wird ausgeführt, wenn FIELDA gleich * ist. Der Vergleichsbefehl nur für diesen Unterblock ist CLI.

>>>>> siehe auch @CAS2, @OFRE und @BEND

@OFRE Rest-Unterblock

Funktion

@OFRE bildet in einer Fallunterscheidung durch Vergleich den Kopf des letzten Unterblocks.

Format

Name	Operation	Operanden
[name]	@OFRE[ST]	

Programmierhinweis

In diesem Unterblock werden alle Fälle behandelt, die in den übrigen Unterblöcken nicht auftraten. Dieser Block sollte verwendet werden für Fehlerfälle, bzw. für Fälle, die bei der Fallunterscheidung irrelevant sind.

>>>>> siehe auch @CAS2 und @OF

@OR Logisches ’Oder’

Funktion

@OR realisiert die logische Oder-Verknüpfung in zusammengesetzten Bedingungen.

Format

Name	Operation	Operanden
[name]	@OR	cond_sym

name Name
cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe 9.2.4 und 9.2.5)

Programmierhinweis

Dem Aufruf des @OR-Makro muß ein Anzeige-setzender Assemblerbefehl folgen (siehe Assemblerbefehle, Beschreibung).

Beispiel

```
Name        Operation   Operanden
            .
            .
            @IF         ZE
            C           R4,ZERO          (01)
            @OR         NZ
            LTR         R5,R5            (02)
            @THEN
            ST          R1,POINT
            @ELSE
            .
            .
            @BEND
            .
            .
```

Der @THEN-Zweig wird ausgeführt, wenn die *erste Bedingung* (01) oder die *zweite Bedingung* (02) zutrifft.

>>>>> siehe auch @AND und @TOR

@PAR Definition von Bereichen

Funktion

@PAR hat drei verschiedene Funktionen. Abhängig vom Format wird für die Parameterübergabe eine Parameterliste aufgebaut oder ein Pseudoabschnitt definiert, oder es wird für die Parameterübernahme ein Pseudoabschnitt definiert.

Format 1: Parameterliste für die statische Parameterübergabe über die STANDARD-Schnittstelle

Name	Operation	Operanden
list_name	@PAR	[PLIST=({name1 | (val1)}[,...]),] VLIST=({name2 | (val2)}[,...]) [,PLEND={YES | NO}]

list_name Name der Parameterliste (63 Zeichen)
name1 Name der Einträge in der Parameterliste
val1 dezimaler selbstdefinierender Wert; erzeugt einen unbenannten Eintrag in der Parameterliste
name2 Name der Felder, deren Adresse in der Parameterliste übergeben werden soll (Aktualparameter)
val2 dezimaler selbstdefinierender Wert; wird direkt in die Parameterliste eingetragen (Aktualparameter)

Beschreibung

Durch den Aufruf von @PAR wird zur Übersetzungszeit eine Parameterliste mit konstanten Werten erzeugt.

PLIST= Gibt den Einträgen der Parameterliste Namen.

Die Angabe von (val1) erzeugt eine namenlose Adreß-Konstante.

VLIST= Gibt die Aktualparameter an, deren Adressen oder Werte in der Parameterliste abgelegt werden.

Wird einem Parameter kein Aktualparameter zugeordnet, muß im VLIST-Operanden an dieser Stelle ein zusätzliches Komma stehen. Eine solche Angabe wird behandelt wie der selbstdefinierende Wert 0.

PLEND= Kann nur in Prozeduren angegeben werden, bei denen ILCS=YES gesetzt ist.
Angabe, ob beim Aufbau der statischen Parameteradreßliste für den letzten Parameter das höchstwertige Bit gesetzt wird.

YES Bit wird gesetzt

NO Bit wird nicht gesetzt

keine Angabe
Bei ILCS=NO: Bit wird aus Kompatibilität gesetzt;
Bei ILCS=YES: Bit wird nicht gesetzt

Die Aktualparameter müssen durch "call by reference" übergeben werden. Selbstdefinierende Parameterwerte dürfen in der Aktualparameterliste nicht übergeben werden (außer es handelt sich um absolute Adressen). Die Aktualparameterliste darf leere Listenelemente enthalten. Diese sind jedoch als nur "Platzhalter" zu betrachten. Vor einem Prozeduraufruf mit der Übergabe dieser Liste (@PASS PAR=<par-list>) müssen freigehaltene Plätze in der Parameterliste vom Benutzer mit korrekten Parameteradreßwerten belegt werden ("call by reference").

Programmierhinweise

1. Im VLIST-Operanden dürfen keine Namen aus Pseudoabschnitten (DSECT-Anweisung, siehe 4.2) angegeben werden.
2. Das höchstwertige Bit der letzten Adreßkonstanten wird auf 1 gesetzt, um das Ende der Parameterliste zu kennzeichnen. Deshalb sollten negative Werte nie direkt übergeben werden.
3. Der Aufruf von @PAR muß zwischen @EXIT und @END der entsprechenden Prozedur erfolgen.
4. @PAR-Aufruf in einer ILCS-Prozedur (@ENTR ILCS=YES):
   - Die Anzahl der Parameter wird in einer neuen EQU-Konstanten abgelegt. Der Name der EQU-Konstanten bestimmt sich aus dem Namen der Parameteradreßliste mit einem am Ende angehängten Zeichen "#".
   - Der Name der Parameterliste darf nicht länger als 63 Zeichen sein, da der Name der zu generierenden EQU-Konstanten genau um ein Zeichen länger ist.

Beispiel

```
Name      | Operation | Operanden
----------+-----------+-----------------------------------------
          | @PASS     | NAME=PROX,PAR=PARLIST
          | .         |
          | .         |
          | @EXIT     |
PARLIST   | @PAR      | PLIST=((1),A,B,C,D),VLIST=(ADR,4,NAME,,FIELD)
          | .         |
          | .         |
```

```
* generierte Instruktionen
PARLIST   | DS        | 0F
          | DC        | A(ADR)
A         | DC        | A(4)
B         | DC        | A(NAME)
C         | DC        | A(0)
D         | DC        | A(FIELD+X'80000000')
```

>>>>> siehe auch @PASS, Format 2

Format 2: Definition eines Pseudoabschnitts im LOCAL-Bereich für die Parameterübernahme

Name	Operation	Operanden
dsect_name	@PAR	D=YES, PLIST=({name / (reg)} [,...]), LEND=YES

dsect_name
: Name des zu generierenden Pseudoabschnitts

name
: Formalparameter; Name eines Feldes, in das beim Aufruf der entsprechende Parameter übernommen wird

reg
: Register, in das beim Aufruf der entsprechende Parameter übernommen wird; positiver absoluter Ausdruck, entweder
  - dezimaler selbstdefinierender Wert oder
  - vordefinierter Name eines Registers oder
  - Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

Beschreibung

Durch den Aufruf von @PAR wird in der gerufenen Prozedur ein Pseudoabschnitt generiert. Dieser gibt die Liste der Formalparameter an, d.h., er beschreibt, bzw. redefiniert die Struktur des LOCAL-Bereichs im Prozedur-STACK der aufgerufenen Prozedur.

Für jeden Formalparameter-Namen aus dem PLIST-Operanden wird ein Eintrag im Pseudoabschnitt erzeugt.

Format 2 von @PAR ist nur in Prozeduren vom Typ E oder I zulässig.

Programmierhinweise

1. Die PLIST-Operanden von @PAR müssen mit den PLIST-Operanden des entsprechenden @ENTR übereinstimmen.
2. Der @PAR-Makro muß nach dem Aufruf von @END der entsprechenden Prozedur folgen.

3. Soll der Pseudoabschnitt außer der Liste der zu übernehmenden Felder und Register noch weitere Daten enthalten, muß er in folgender Weise erklärt werden:

```
Name          | Operation | Operanden
              |     .     |
              |     .     |
                                    ⎡name ⎤
dsect_name    | @PAR      | D=YES,PLIST=(⎨     ⎬[,...])
                                    ⎣(reg)⎦
...           | DS        | ...
              |     .     |
              |     .     |
dsect_name    | @PAR      | LEND=YES
```

Beispiel

Das Beispiel zeigt die Generierung einer DSECT im LOCAL-Bereich einer gerufenen Prozedur.

```
Name          | Operation | Operanden
PRO           | @ENTR     | TYP=E,LOCAL=IN,PLIST=(INPAR1,INPAR2,(R5),INPAR3)
              |     .     |
              |     .     |
              | @END      |
IN            | @PAR      | D=YES,LEND=YES,PLIST=(INPAR1,INPAR2,(R5),INPAR3)
              |     .     |
              |     .     |

* generierte Instruktionen
IN            | DSECT     |
              | ORG       | *+96
INPAR1        | DS        | A
INPAR2        | DS        | A
INPAR3        | DS        | A
LIN           | EQU       | *-IN
PRO           | CSECT     |
```

>>>>> siehe auch @ENTR und @PASS, Format 3

Format 3: Definition des Pseudoabschnitts für den LOCAL-Bereich

Name	Operation	Operanden
dsect_name	@PAR	{D=YES } {LEND=YES}

dsect_name
Name des zu generierenden Pseudoabschnitts

Beschreibung

D=YES @PAR mit diesem Operanden kennzeichnet den Beginn des Pseudoabschnitts.

LEND=YES
@PAR mit diesem Operanden kennzeichnet das Ende des Pseudoabschnitts.

Zwischen @PAR D=YES und @PAR LEND=YES müssen Datenfelder definiert sein. Diese Folge von Instruktionen kann verwendet werden, um die Struktur eines lokalen Datenbereichs im Prozedur-STACK zu beschreiben, der durch @ENTR..., LOCAL=dsect_name angefordert wurde.

Programmierhinweis

Der Name des zu generierenden Pseudoabschnitts muß mit dem LOCAL-Operanden von @ENTR der entsprechenden Prozedur übereinstimmen.

Beispiel

```
Name        Operation   Operanden
PRO         @ENTR       TYP=I,LOCAL=DUMMY
            .
            .
            @END
DUMMY       @PAR        D=YES
NAME1       DS          ..
            .
            .
DUMMY       @PAR        LEND=YES
```

>>>>> siehe auch @ENTR

@PASS Prozedur-Aufruf

Funktion

@PASS realisiert den Aufruf einer Unter-Prozedur. Die aufrufende Prozedur kann dabei Parameter an die aufgerufene Prozedur übergeben.

Format 1: Aufruf ohne Parameterübergabe

Name	Operation	Operanden
[name1]	@PASS	NAME=int_name EXTNAME=ext_name ADDR=name2 ADDR=(reg)

name1 Name

int_name Name einer Prozedur im gleichen Modul wie die rufende (interne Prozedur)

ext_name Name einer Prozedur in einem anderen Modul als die rufende (externe Prozedur)

name2 Name eines Wortes, das die Adresse der gerufenen Prozedur enthält

reg Register, das die Adresse der gerufenen Prozedur enthält; positiver absoluter Ausdruck; entweder

- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

Programmierhinweise

1. Eine mit EXTNAME=ext_name aufgerufene Prozedur muß mit @ENTR TYP=E gekennzeichnet sein.
2. Bei der Angabe von ADDR= muß name2, bzw. reg in der Prozedur mit der Adresse der gerufenen Prozedur versorgt werden.

Format 2: Aufruf mit statischer Parameterübergabe über die STANDARD-Schnittstelle

Name	Operation	Operanden
[name1]	@PASS	{NAME=int_name / EXTNAME=ext_name / ADDR={name2 / (reg)}},PAR={list_name / (list_reg)}

name1 Name

int_name Name einer Prozedur im gleichen Modul wie die rufende (interne Prozedur)

ext_name Name einer Prozedur in einem anderen Modul als die rufende (externe Prozedur)

name2 Name eines Wortes, das die Adresse der gerufenen Prozedur enthält

reg Register, das die Adresse der gerufenen Prozedur enthält; positiver absoluter Ausdruck; entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

list_name Name der Parameterliste

list_reg Nur in Prozeduren zulässig, in denen ILCS=NO gesetzt ist.
Register, das die Adresse der Parameterliste enthalten muß; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde.

Beschreibung

list_name, bzw. list_reg gibt den Namen, bzw. die Adresse der Parameterliste an, die in der rufenden Prozedur durch den @PAR-Makro erzeugt wurde.

Die Adresse der Parameterliste wird über das Register 1 an die aufgerufene Prozedur übergeben.

Bei ILCS=YES wird Register 0 mit der Anzahl der Parameter geladen.

Die Verwendung von Register 1 im Operanden ADDR=reg ist unzulässig.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------------------------------------
PRO1      | @ENTR     | TYP=M
          | .         |
          | .         |
          | @PASS     | NAME=PRO2,PAR=PARLIST
          | .         |
          | .         |
          | @EXIT     |
PARLIST   | @PAR      | PLIST=(PAR1,PAR2,PAR3),VLIST=(FIELD,27,SET)
          | @END      |
          | .         |
          | .         |
PRO2      | @ENTR     | TYP=I,LOCAL=IN,PLIST=(INPAR1,INPAR2,INPAR3)
          | .         |
          | .         |
          | @EXIT     |
          | @END      |
IN        | @PAR      | D=YES,LEND=YES,PLIST=(INPAR1,INPAR2,INPAR3)
          | .         |
          | .         |
```

>>>>> siehe auch @PAR, Format 1

Format 3: Aufruf mit dynamischer Parameterübergabe; STANDARD- oder OPTIMAL-Schnittstelle

Name	Operation	Operanden
[name1]	@PASS	{NAME=int_name / EXTNAME=ext_name / ADDR={name2 / (reg)}} ,PLIST=({par_name / (par_reg)}[,...]) [,PASS={STA[NDARD] / OPT[IMAL]}] [,PLEND={YES / NO}]

name1 Name

int_name Name einer Prozedur im gleichen Modul wie die rufende (interne Prozedur)

ext_name Name einer Prozedur in einem anderen Modul als die rufende (externe Prozedur)

name2 Name eines Wortes, das die Adresse der gerufenen Prozedur enthält

reg Register, das die Adresse der gerufenen Prozedur enthält; positiver absoluter Ausdruck; entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde

par_name Name des Feldes, dessen Adresse an die aufgerufene Prozedur übergeben werden soll

par_reg Register, dessen Inhalt an die aufgerufene Prozedur übergeben werden soll; positiver absoluter Ausdruck, entweder
- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde.

Bei ILCS=YES muß das Register die Adresse des zu übergebenden Wertes enthalten ("call by reference").

Beschreibung

PASS=STA

Die Parameterübergabe erfolgt über die STANDARD-Schnittstelle.

Der PLIST-Operand gibt die Aktualparameter an. Im LOCAL-Bereich der rufenden Prozedur wird eine Parameterliste mit den Aktualparametern angelegt.

Die Verwendung von Register 1 im Operanden ADDR=reg ist nicht zulässig.

PASS=OPT

Nur bei ILCS=NO zulässig.
Die Parameterübergabe erfolgt über die OPTIMAL-Schnittstelle.

Der PLIST-Operand gibt die Aktualparameter an. Bis zu vier Parameter werden in den Registern 1 bis 4 übergeben. Bei mehr als vier Parametern werden die ersten drei in den Registern 2 bis 4 übergeben. Für die restlichen wird eine Parameterliste angelegt, deren Adresse in Register 1 übergeben wird.

Die Verwendung der Register 1 bis 4 im Operanden ADDR=reg ist nicht zulässig.

PLEND= Kann nur in Prozeduren angegeben werden, bei denen ILCS=YES gesetzt ist und nur bei Angabe einer PLIST.
Angabe, ob bei der dynamischen Parameterübergabe in der Parameteradreßliste für den letzten Parameter das höchswertige Bit gesetzt wird.

YES Bit wird gesetzt

NO Bit wird nicht gesetzt

keine Angabe
Bei ILCS=NO: Bit wird aus Kompatibilität gesetzt;
Bei ILCS=YES: Bit wird nicht gesetzt

Bei ILCS=YES wird Register 0 mit der Anzahl der Parameter geladen.

Register 1 wird mit der Adresse einer Parameteradreßliste geladen.

Das höchstwertige Bit bei der Adresse des letzten Parameters in der Parameteradreßliste wird abhängig vom Parameter PLEND gesetzt.

Werden keine Parameter übergeben, so werden bei ILCS=YES die Register 0 und 1 auf Null gesetzt.

Format 3 von @PASS ist nur in Prozeduren vom Typ M, E oder I zulässig.

Programmierhinweis

Im Falle eines @PASS-Aufrufs aus einer ILCS-Prozedur ist der Anwender selbst für das korrekte Belegen der Parameteradreßliste verantwortlich: ILCS verlangt immer "call by reference". Bei Nicht-ILCS ist auch "call by value" möglich. Eine Überprüfung durch die Makros zum Übersetzungszeitpunkt ist z.B. wegen EQU-Symbolen nicht möglich.

Beispiel

Das Beispiel zeigt die Übergabe von drei Parametern über die STANDARD-Schnittstelle.

```
Name        Operation   Operanden
PRO1        @ENTR       TYP=M,MAXPRM=3
            @DATA       CLASS=A,BASE=R6,DSECT=DUMMY
            .
            .
            @PASS       NAME=PRO2,PLIST=(FIELD1,(R7),FIELD4)
            .
            .
            @EXIT
            @END
DUMMY       DSECT
FIELD1      DS          CL10
            .
            .
FIELD4      DS          CL25
LDUMMY      EQU         *-DUMMY
            .
            .
PRO2        @ENTR       TYP=I,LOCAL=IN,PLIST=(INPAR1,INPAR2,INPAR3)
            .
            .
            @EXIT
            @END
IN          @PAR        D=YES,LEND=YES,PLIST=(INPAR1,INPAR2,INPAR3)
            .
            .
```

>>>>> siehe auch @ENTR und @PAR, Format 2

@SETJV Monitorjobvariable setzen

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@SETJV (**SET J**ob **V**ariable) setzt einen Wert in das MONJV-Feld der PCD (siehe auch 9.6.7).

Format

Name	Operation	Operanden
[name]	@SETJV	[MONJV={monjv / symb Adr / (reg)}]

name Name
monjv alphanumerischer Wert
symb Adr symbolische Adresse eines Feldes
reg Mehrzweckregister, enthält die Adresse des Feldes.

Beschreibung

MONJV= definiert den neuen MONJV-Wert

monjv alphanumerischer Wert der Länge 4, eingeschlossen in Hochkommas.

symb Adr1
symbolische Adresse eines Feldes mit dem String.

(reg1) wahlweise Register, das die Adresse des Strings enthält.

keine Angabe
In das PCD-Feld MONJV werden vier Blanks eingetragen.

Programmierhinweise

1. Ist die Zeichenfolge des MONJV-Wertes länger als 4 Zeichen, so werden die ersten 4 Zeichen verwendet.
2. Ist die Zeichenfolge des MONJV-Wertes kürzer als 4 Zeichen, so wird rechts mit Blanks aufgefüllt.

@SETPM Programmaske setzen oder rücksetzen

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@SETPM (**SET P**rogram **M**ask) setzt die Programmaske oder setzt eine veränderte Programmaske zurück (siehe auch 9.6.6).

Format

Name	Operation	Operanden
[name]	@SETPM	[PMASK={symb Adr / (reg)}]

name Name
symb Adr symbolische Adresse eines Feldes
reg Mehrzweckregister, enthält die Programmaske.

Beschreibung

PMASK= definiert die neue Programmaske

symb Adr
: symbolische Adresse eines Feldes der Länge 1 mit der neuen Programmmaske in hexadezimaler Form.

(reg) wahlweise Register mit der neuen Programmaske im linken Byte.

keine Angabe
: Die Programmaske wird auf den ILCS-Standardwert gesetzt.

Programmierhinweis

Wurde die Programmaske verändert, so muß die Maske zurückgesetzt werden vor einem @EXIT bzw. immer dann, wenn eine ILCS-Prozedur nach ILCS-Konventionen aufgerufen wird. Die Programmaske muß den ILCS-Standardwert haben.

@STXDI STXIT-Behandlungsroutine abmelden

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@STXDI (**STX**IT **di**sable) meldet eine STXIT-Behandlungsroutine ab (siehe auch 9.6.5, Unterbrechungsereignisse behandeln).

Format

Name	Operation	Operanden
[name]	@STXDI	STXID={symb Adr / (reg)}

name Name
symb Adr symbolische Adresse eines Feldes
reg Mehrzweckregister, enthält die Adresse der Kurzkennung

Beschreibung

STXID Kurzkennung der STXIT-Routine
Die Kurzkennung wird vom Makro @STXEN geliefert.

symb Adr
symbolische Adresse eines Feldes der Länge 4 auf Wortgrenze mit der Kurzkennung.

(reg) wahlweise Register mit der Adresse der Kurzkennung.

Der Inlinecode des Makros übernimmt die Angaben aus dem Aufruf in einen ILCS-konformen Parameterblock; mit dieser Versorgung wird der entsprechende Entry im Standard-STXIT-Handler zum Abmelden der STXIT-Behandlungsroutine angesprungen.

Nach Rückkehr des Standard-STXIT-Handlers in die aufrufende Prozedur enthält Register 15 einen Returncode, der besagt, ob die Funktion ausgeführt wurde oder nicht bzw. welche Fehler auftraten.

@STXEN STXIT-Behandlungsroutine anmelden

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@STXEN (**STX**IT **en**able) meldet eine STXIT-Behandlungsroutine an (siehe auch 9.6.5, Unterbrechungsereignisse behandeln).

Format

Name	Operation	Operanden
[name]	@STXEN	STXNAME={name1 | symb Adr1 | (reg1)} STXID={symb Adr2 | (reg2)} STXADR={symb Adr3 | (reg3)} [STXSVC={symb Adr4 | (reg4)}]

name Name
name1 alphabetische Zeichenfolge

symb Adr1...symb Adr4
symbolische Adressen von Feldern

reg1...reg4
Mehrzweckregister, in denen die Adresse eines Feldes steht.

Beschreibung

STXNAME= Name des STXIT-Ereignisses

name1 ’String’ mit max. 7 Zeichen.
Folgende Werte können für die entsprechenden Unterbrechungsklassen angegeben werden:

```
’PROCHK ’    "Programmüberprüfung"
’TIMER  ’    "Intervallzeitgeber CPU"
’RUNOUT ’    "Ende Programmlaufzeit"
’ERROR  ’    "Nicht behebbarer Programmfehler"
’ABEND  ’    Klasse "ABEND"
’ESCPBRK’    Klasse "ESCPBRK"
’TERM   ’    "Programmbeendigung"
’RTIMER ’    "Intervallzeitgeber Real-Zeit"
’INTR’       "Mitteilung an das Programm"
’MSG    ’    "Mitteilung an das Programm"
’HWERROR’    "Hardwarefehler"
’SVC    ’    "SVC-Unterbrechung"
```

symb Adr1
symbolische Adresse eines Feldes der Länge 7 mit einem der oben genannten Strings.

(reg1) wahlweise Register, das die Adresse des Strings enthält.

STXID= zurückgemeldete Kurzkennung

symb Adr2
symbolische Adresse eines Feldes der Länge 4 auf Wortgrenze.

(reg2) wahlweise Register mit der Adresse des Rückmeldefeldes.

STXADR= Startadresse der STXIT-Routine

symb Adr3
symbolische Adresse eines Feldes der Länge 4 auf Wortgrenze mit der Startadresse.

(reg3) wahlweise Register mit der Startadresse.

STXSVC= Liste mit SVC-Nummern
Der Parameter muß angegeben werden, wenn im ersten Parameter als Ereignisname ’SVC’ angegeben wurde.

symb Adr4
symbolische Adresse einer Liste auf Halbwortgrenze mit Anzahl der Einträge und SVC-Nummern.

(reg4) wahlweise Register mit der SVC-Listenadresse.

Der Inlinecode des Makros übernimmt die Angaben aus dem Aufruf in einen ILCS-konformen Parameterblock; mit dieser Versorgung wird der entsprechende Entry im Standard-STXIT-Handler zum Anmelden der STXIT-Behandlungsroutine angesprungen.

Nach Rückkehr des Standard-STXIT-Handlers in die aufrufende Prozedur enthält Register 15 einen Returncode, der besagt, ob die Funktion ausgeführt wurde oder nicht bzw. welche Fehler auftraten.

Programmierhinweis

Die Ereignisklassen ’HWERROR’ und ’MSG’ werden erst ab BS2000 V11.0 unterstützt (auch abhängig von der ILCS-Version).
Die Angabe ’MSG’ wird schon jetzt vom Makro auf ’INTR’ abgebildet.

@STXIM Layout der Unterbrechungsmeldung definieren

Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

Funktion

@STXIM (**STX**IT **I**nterrupt **M**essage) definiert das Layout der Parameter, die vom Standard-STXIT-Handler (SSH) an die Benutzerroutine übergeben werden.

Format

Name	Operation	Operanden
[name]	@STXIM	[pre]=zeichenfolge

name Name
pre Präfix
zeichenfolge
alphanumerische Zeichen

Beschreibung

pre= Präfix für die symbolischen Namen der Einzelfelder

zeichenfolge
max. 4 Zeichen
pre muß der Assemblersyntax für Namen genügen.

Mit dem Präfix werden folgende DS-Anweisungen für die Einzelfelder definiert:

```
preSTIW   DS   F     STXIT Interrupt Weight
preSTMG   DS   CL64  Interrupt Message
```

Diese Parameter werden vom Standard-STXIT-Handler bei Eintreten eines STXIT-Ereignisses an die Benutzerroutine übergeben.

Programmierhinweis

Ein Präfix mit mehr als 4 Zeichen wird auf 4 Zeichen gekürzt.

@THEN Ja-Unterblock

Funktion

@THEN bildet den Kopf des Ja-Unterblocks in einer Entscheidung.

Format

Name	Operation	Operanden
[name]	@THEN	

Beschreibung

Der Unterblock in einer @IF-Verzweigung, der mit @THEN beginnt, wird **nur** durchlaufen, wenn die mit @IF gesetzte Bedingung zutrifft.

>>>>> siehe auch @IF, @ELSE und @BEND

@THRU Iterative Schleife

Funktion

@THRU bildet den Kopf einer Schleifenkonstruktion. Die Anzahl der Durchläufe wird durch die Angabe von Anfangswert und Endwert einer Laufvariablen bestimmt.

Format

Name	Operation	Operanden
[name]	@THRU	(reg1),{(reg2) / name2 / literal2}[,{(reg3) / name3 / literal3}]

name Name

reg1, reg2, reg3
: Mehrzweckregister; positive absolute Ausdrücke, entweder
  - dezimale selbstdefinierende Werte oder
  - vordefinierte Namen von Registern oder
  - Namen, denen ein entsprechender selbstdefinierender Wert zugewiesen wurde

name2, name3
: Namen von Konstanten oder Speicherbereichen, die auf Halbwortgrenze ausgerichtet und jeweils ein Halbwort lang sein müssen

literal2, literal3
: Literale, die auf Halbwortgrenze ausgerichtet und jeweils ein Halbwort lang sein müssen.

Beschreibung

reg1 enthält den Anfangswert für die Laufvariable

reg2, bzw. name2 oder literal2
: gibt den Endwert für die Laufvariable an

reg3 bzw. name3 oder literal3
: gibt die Schrittweite an.
  Standardwert für die Schrittweite ist 1

Die Schrittweite wird jeweils nach Ausführung des Schleifen-Unterblocks zur Laufvariablen addiert. Ist die Summe aus Laufvariable und Schrittweite größer als der Endwert, wird der Strukturblock verlassen.

Programmierhinweise

1. reg1 muß vor dem Aufruf des @THRU-Makro mit dem Anfangswert geladen werden.
2. Wird keine Schrittweite angegeben, darf Register 0 nicht für den Anfangswert verwendet werden.
3. Ein @THRU-Makro mit Anfangswert ≤ Endwert und Schrittweite ≤ 0 führt zu einer Endlosschleife.

Beispiel

```
Name      | Operation | Operanden
----------|-----------|-----------------
          |     .     |
          |     .     |
          |     LH    |  R6,BEGL
LOOP      |     @THRU |  (R6),ENDL
          |     @DO   |
          |     .     |
          |     .     |
          |     @BEND |
          |     .     |
          |     .     |
BEGL      |     DC    |  H'1'
ENDL      |     DC    |  H'10'
```

In diesem Beispiel ist der Anfangswert der Laufvariablen 1, der Endwert 10 und die Schrittweite ebenfalls 1 (Standardwert). Daher wird der Schleifenunterblock 10-mal durchlaufen.

>>>>> siehe auch @DO und @BEND

@TOR Logisches 'Oder mit Priorität'

Funktion

@TOR realisiert in zusammengesetzten Bedingungen eine logische Oder-Verknüpfung, die aber eine höhere Bindungspriorität hat als @AND.

Format

Name	Operation	Operanden
[name]	@TOR	cond_sym

name Name
cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe 9.2.4 und 9.2.5)

Programmierhinweise

Dem Aufruf des @TOR-Makro muß ein Anzeige-setzender Assemblerbefehl folgen (siehe Assemblerbefehle, Beschreibung).

Beispiel

```
Name        Operation   Operanden
LOOP        @CYCL
            .
            .
            @WHEN       EQ
            CR          R1,R2
            @AND        EQ
            CR          R7,R8
            @TOR        EQ
            CLI         FIELD,C'3'
            @BREA
            .
            .
            @BEND
```

Die Schleife wird verlassen, entweder bei
- R1 = R2 und R7 = R8 oder bei
- R1 = R2 und FIELD = 3.

>>>>> siehe auch @AND und @OR

@WHEN Bedingung für Schleifenabbruch

Funktion

@WHEN gibt in einer Schleife mit freier Endebedingung oder in einer Zählschleife mit freier Endebedingung die Bedingung an, die zum Abbrechen der Schleife führt.

Format

Name	Operation	Operanden
[name]	@WHEN	cond_sym

name Name
cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe 9.2.4 und 9.2.5)

Beschreibung

cond_sym legt die Bedingung (siehe 9.2.4 und 9.2.5) fest, die zum Beenden der entsprechenden Schleife führen soll. Trifft die Bedingung zu, wird die Schleife mit dem @BREA-Makro verlassen.

Beispiel siehe bei @BREA

>>>>> siehe auch @BREA, @CYCL und @BEND

@WHIL Schleife mit Vorabprüfung

Funktion

@WHIL bildet den Kopf einer Schleifenkonstruktion, bei der die Anzahl der Durchläufe vom Zutreffen einer Bedingung bestimmt wird.

Format

Name	Operation	Operanden
[name]	@WHIL[E]	cond_sym

name Name
cond_sym vordefiniertes oder benutzereigenes Bedingungssymbol (siehe 9.2.4 und 9.2.5)

Beschreibung

cond_sym legt die Bedingung (siehe 9.2.4 und 9.2.5) fest, die zum Durchlaufen des Schleifenunterblocks führen soll. Beim Aufruf des @WHIL-Makro wird die Bedingung geprüft und wenn sie zutrifft, der Schleifenunterblock ausgeführt.

Beispiel

Name	Operation	Operanden
LOOP	@WHIL	EQ
	CLI	END,C'N'
	@DO	
	.	
	.	
	@BEND	

Der Schleifenunterblock wird ausgeführt, solange END gleich N ist.

>>>>> siehe auch @DO und @BEND

11 Anhang

11.1 Zusammenfassung der DC-Konstanten

Typ	Spezifiziert durch	Ausrichtung	implizite Länge (byte)	Längen-faktor	Exponenten-faktor	Skalen-faktor	Auffüllen, Abschneiden
A	absoluter oder relativer Ausdruck	Wort	4	1 bis 4			links
B	Binär-Ziffern	Byte	wie benötigt	1 bis 256			links
C	Zeichen	Byte	wie benötigt	1 bis 256			rechts
D	Dezimal-ziffern	Doppel-wort	8	1 bis 8	-85 bis +75	0 bis 14	Auffüllen rechts Abschneiden nicht anwendbar
E	Dezimal-ziffern	Wort	4	1 bis 8	-85 bis +75	0 bis 14	Auffüllen rechts Abschneiden nicht anwendbar
F	Dezimal-ziffern	Wort	4	1 bis 8	-85 bis +75	-187 bis +346	links
H	Dezimal-ziffern	Halbwort	2	1 bis 8	-85 bis +75	-187 bis +346	links
L	Dezimal-ziffern	Doppel-wort	16	1 bis 16	-85 bis +75	0 bis 28	Auffüllen rechts Abschneiden nicht anwendbar
P	Dezimal-ziffern	Byte	wie benötigt	1 bis 16			links
Q	Name eines Pseudo-registers	Wort	4	1 bis 4		links	
S	symbolische oder nicht-symbolische Adresse	Halbwort	2	2			
V	Name	Wort	4	3 oder 4			links
X	Sedezimal-Ziffern	Byte	wie benötigt	1 bis 256			links
Y	absoluter oder relativer Ausdruck	Halbwort	2	1 oder 2		links	
Z	Dezimal-ziffern	Byte	wie benötigt	1 bis 16			links

11.2 Format der Assemblerbefehle

In dieser Befehlsmenge sind die Befehle der Befehlssätze BS2000-NXS (SET1), BS2000-XS (SET3) und BS2000-ESA enthalten (die Assemblerbefehle sind in der Sprachbeschreibung "Assemblerbefehle" [3] beschrieben).
Der Befehlssatz BS2000-NXS unterstützt die Zentraleinheiten mit der 24-Bit-Adressierung (NXS bedeutet Nicht eXtended System).
Der Befehlssatz BS2000-XS unterstützt die XS-Anlagen mit der 31-Bit-Adressierung (XS bedeutet eXtended System).
Der Befehlssatz BS2000-ESA unterstützt die ESA-Anlagen, auf denen eine Erweiterung der virtuellen Adreßräume möglich ist (ESA bedeutet Enterprise Systems Architecture).
Im Befehlssatz BS2000-XS ist der Befehlssatz BS2000-NXS enthalten und beide sind im Befehlssatz BS2000-ESA enthalten.
In der Spalte NXS / XS / ESA ist jeweils mit dem Anfangsbuchstaben N, X oder E markiert, zu welchem Befehlssatz der Befehl gehört.
In folgender Darstellung sind die mit N markierten Befehle eine Befehlsgrundmenge und die mit X oder E markierten Befehle die zusätzliche Menge dieser Befehlssätze.

Mnem. Code	Befehlsname	NXS XS ESA	Masch. Code	Länge	Operandenformat
A	Addieren	N	5A	4	R1,D2(X2,B2)
AD	Addieren normalisiert lang	N	6A	4	R1,D2(X2,B2)
ADR	Addieren normalisiert lang	N	2A	2	R1,R2
AE	Addieren normalisiert kurz	N	7A	4	R1,D2(X2,B2)
AER	Addieren normalisiert kurz	N	3A	2	R1,R2
AH	Addieren Halbwort	N	4A	4	R1,D2(X2,B2)
AL	Addieren ohne Vorzeichen	N	5E	4	R1,D2(X2,B2)
ALR	Addieren ohne Vorzeichen	N	1E	2	R1,R2
AP	Addieren dezimal	N	FA	6	D1(L1,B1),D2(L2,B2)
AR	Addieren	N	1A	2	R1,R2
AU	Addieren nicht normalisiert kurz	N	7E	4	R1,D2(X2,B2)
AUR	Addieren nicht normalisiert kurz	N	3E	2	R1,R2
AW	Addieren nicht normalisiert lang	N	6E	4	R1,D2(X2,B2)
AWR	Addieren nicht normalisiert lang	N	2E	2	R1,R2
AXR	Addieren normalisiert mit erweiterter Länge	N	36	2	R1,R2
BAL	Springen und Speichern Rücksprungadresse	N	45	4	R1,D2(X2,B2)
BALR	Springen und Speichern Rücksprungadresse	N	05	2	R1,R2
BAS	Springen und Speichern Rücksprungadresse	N	4D	4	R1,D2(X2,B2)
BASR	Springen und Speichern Rücksprungadresse	N	0D	2	R1,R2
BASSM	Branch and Save and Set Mode	X	0C	2	R1,R2
BC	Springen bedingt	N	47	4	I,D2(X2,B2)
BCR	Springen bedingt	N	07	2	I,R2
BCT	Springen nach Zählen	N	46	4	R1,D2(X2,B2)
BCTR	Springen nach Zählen	N	06	2	R1,R2
BSM	Branch and Save	X	0B	2	R1,R2
BXH	Springen wenn Index größer	N	86	4	R1,R3,D2(B2)
BXLE	Springen wenn Index kleiner gleich	N	87	4	R1,R3,D2(B2)
C	Vergleichen algebraisch	N	59	4	R1,D2(X2,B2)
* CCPU	Prüfen Zentraleinheit	N	AC	4	D1(B1),I2
CCW	Define Channel Command Word	N		8	I1,I2,I3,I4
CCW0	Define Channel Command Word (Format 0)	X		8	I1,I2,I3,I4
CCW1	Define Channel Command Word (Format 1)	X		8	I1,I2,I3,I4
CD	Vergleichen lang	N	69	4	R1,D2(X2,B2)
CDR	Vergleichen lang	N	29	2	R1,R2
CDS	Vergleichen doppelt und austauschen	N	BB	4	R1,R3,D2(B2)
CE	Vergleichen kurz	N	79	4	R1,D2(X2,B2)
CER	Vergleichen kurz	N	39	2	R1,R2
CH	Vergleichen Halbwort	N	49	4	R1,D2(C2,B2)
* CIOC	Prüfen Ein-Ausgabesteuerung	N	AD	4	D1(B1),I2
* CKC	Prüfen Kanal	N	9F	4	D1(B1)
CL	Vergleichen logisch	N	55	4	R1,D2(X2,B2)
CLC	Vergleichen logisch	N	D5	6	D1(L,B1),D2(B2)
CLCL	Vergleichen logisch lang	N	0F	2	R1,R2

Mnem. Code	Befehlsname	NXS XS ESA	Masch. Code	Länge	Operandenformat
CLI	Vergleichen logisch	N	95	4	D1(B1),I2
CLM	Vergleichen logisch mit Maske	N	BD	4	R1,M3,D2(B2)
CLR	Vergleichen logisch	N	15	2	R1,R2
CP	Vergleichen dezimal	N	F9	6	D1(L1,B1),D2(L2,B2)
CPYA	Copy Access Register	E	B24D	4	R1,R2
CR	Vergleichen algebraisch	N	19	2	R1,R2
CS	Vergleichen und austauschen	N	BA	4	R1,R3,D2(B2)
* CSCH	Clear Subchannel	X	B230	4	kein Operand
CVB	Umwandeln in Binärform	N	4F	4	R1,D2(X2,B2)
CVD	Umwandeln in Dezimalform	N	4E	4	R1,D2(X2,B2)
D	Dividieren	N	5D	4	R1,D2(X2,B2)
DD	Dividieren lang	N	6D	4	R1,D2(X2,B2)
DDR	Dividieren lang	N	2D	2	R1,R2
DE	Dividieren kurz	N	7D	4	R1,D2(X2,B2)
DER	Dividieren kurz	N	3D	2	R1,R2
* DIG	Suchen Maschinenfehler	N	83	4	D1(B1)
DP	Dividieren dezimal	N	FD	6	D1(L1,B1),D2(L2,B2)
DR	Dividieren	N	1D	2	R1,R2
DXR	Divide extended	X	B22D	4	R1,R2
EAR	Extract Access Register	E	B24F	4	R1,R2
ED	Aufbereiten	N	DE	6	D1(L,B1),D2(B2)
EDMK	Aufbereiten und Markieren	N	DF	6	D1(L,B1),D2(B2)
** EPAR	Extract Primary ASN	X	B226	4	R1
** ESAR	Extract Secondary ASN	X	B227	4	R1
EX	Ausführen	N	44	4	R1,D2(X2,B2)
* FC	Ausführen Sonderfunktionen	N	9A	4	D1(B1),I2
* FCAL	Ausführen Sonderfunktionen	N	B7	4	D1(B1),I2
HDR	Halbieren lang	N	24	2	R1,R2
* HDV	Halten Gerät	N	9E	4	D1(B1)
HER	Halbieren kurz	N	34	2	R1,R2
* HSCH	Halt Subchannel	X	B231	4	kein Operand
** IAC	Insert Address Space Control	E	B224	4	R1
IC	Laden Zeichen	N	43	4	R1,D2(X2,B2)
ICM	Einsetzen Zeichen mit Maske	N	BF	4	R1,M3,D2(B2)
* IDL	Warten	N	80	4	I2
** IPK	Insert PSW Key	X	B20B	4	kein Operand
IPM	Einsetzen Programmaske	N	B222	4	R1
* ISK	Abfragen Speicherschlüssel	N	09	2	R1,R2
** IVSK	Insert Virtual Storage Key	X	B223	4	R1,R2
L	Laden	N	58	4	R1,D2(X2,B2)
LA	Laden Adresse	N	41	4	R1,D2(X2,B2)
LAE	Load Address Extended	E	51	4	R1,D2(X2,B2)
LAM	Load Access Multiple	E	9A	4	R1,R3,D2(B2)
LCDR	Laden Komplement lang	N	23	2	R1,R2
LCER	Laden Komplement kurz	N	33	2	R1,R2
LCR	Laden Komplement	N	13	2	R1,R2
LD	Laden lang	N	68	4	R1,D2(X2,B2)
LDR	Laden lang	N	28	2	R1,R2
LE	Laden kurz	N	78	4	R1,D2(X2,B2)
LER	Laden kurz	N	38	2	R1,R2
LH	Laden Halbwort	N	48	4	R1,D2(X2,B2)
LM	Laden mehrfach	N	98	4	R1,R3,D2(B2)

Mnem. Code	Befehlsname	NXS XS ESA	Masch. Code	Länge	Operandenformat
LNDR	Laden negativ lang	N	21	2	R1,R2
LNER	Laden negativ kurz	N	31	2	R1,R2
LNR	Laden negativ	N	11	2	R1,R2
LPDR	Laden positiv lang	N	20	2	R1,R2
LPER	Laden positiv kurz	N	30	2	R1,R2
LPR	Laden positiv	N	10	2	R1,R2
LR	Laden	N	18	2	R1,R2
LRDR	Laden und Runden mit erweiterter Länge, Ergebnis lang	N	25	2	R1,R2
LRER	Laden und Runden mit erweiterter Länge, Ergebnis kurz	N	35	2	R1,R2
* LSM	Laden Schattenspeicher	N	D9	6	D1(L,B1),D2(B2)
* LSP	Laden Zwischenspeicher	N	D8	6	D1(L,B1),D2(B2)
LTDR	Laden und Testen lang	N	22	2	R1,R2
LTER	Laden und Testen kurz	N	32	2	R1,R2
LTR	Laden und Testen	N	12	2	R1,R2
M	Multiplizieren	N	5C	4	R1,D2(X2,B2)
MD	Multiplizieren lang	N	6C	4	R1,D2(X2,B2)
MDR	Multiplizieren lang	N	2C	2	R1,R2
ME	Multiplizieren kurz	N	7C	4	R1,D2(X2,B2)
MER	Multiplizieren kurz	N	3C	2	R1,R2
MH	Multiplizieren Halbwort	N	4C	4	R1,D2(X2,B2)
MP	Multiplizieren dezimal	N	FC	6	D1(L1,B1),D2(L2,B2)
MR	Multiplizieren	N	1C	2	R1,R2
* MSCH	Modify Subchannel	X	B232	4	D2(B2)
MVC	Übertragen Zeichenfolge	N	D2	6	D1(L,B1),D2(B2)
MVCL	Übertragen Zeichenfolge lang	N	0E	2	R1,R2
** MVCP	Move to Primary	X	DA	6	D1(R1,B1),D2(B2),R3
** MVCS	Move to Secondary	X	DB	6	D1(R1,B1),D2(B2),R3
MVI	Ersetzen Zeichen	N	92	4	D1(B1),I2
MVN	Übertragen numerisch	N	D1	6	D1(L,B1),D2(B2)
MVO	Übertragen mit Versetzen	N	F1	6	D1(L1,B1),D2(L2,B2)
MVZ	Übertragen Zonen	N	D3	6	D1(L,B1),D2(B2)
MXD	Multiplizieren mit erweiterter Länge, Ergebnis lang	N	67	4	R1,D2(X2,B2)
MXDR	Multiplizieren mit erweiterter Länge, Ergebnis lang	N	27	2	R1,R2
MXR	Multiplizieren mit erweiterter Länge, Ergebnis kurz	N	26	2	R1,R2
N	UND	N	54	4	R1,D2(X2,B2)
NC	UND	N	D4	6	D1(L,B1),D2(B2)
NI	UND	N	94	4	D1(B1),I2
NR	UND	N	14	2	R1,R2
O	ODER	N	56	4	R1,D2(X2,B2)
OC	ODER	N	D6	6	D1(L,B1),D2(B2)
OI	ODER	N	96	4	D1(B1),I2
OR	ODER	N	16	2	R1,R2
PACK	Packen	N	F2	6	D1(L1,B1),D2(L2,B2)
** PC	Wechseln Funktionszustand	X	B218	4	D2(B2)
** PT	Program Transfer	X	B228	4	R1,R2
* RCHP	Reset Channel Path	X	B23B	4	kein Operand
* RDD	Lesen direkt	N	85	4	D1(B1),I2

Mnem. Code	Befehlsname	NXS XS ESA	Masch. Code	Länge	Operandenformat
* RSCH	Resume Subchannel	X	B238	4	kein Operand
S	Subtrahieren	N	5B	4	R1,D2(X2,B2)
SAC	Set Address Space Control	E	B219	4	D2(B2)
* SAL	Set Address Limit	X	B237	4	kein Operand
SAR	Set Access Register	E	B24E	4	R1,R2
* SCHM	Set Channel Monitor	X	B23C	4	kein Operand
SD	Subtrahieren normalisiert lang	N	6B	4	R1,D2(X2,B2)
SDR	Subtrahieren normalisiert lang	N	2B	2	R1,R2
* SDV	Starten Gerät	N	9C	4	D1(B1)
SE	Subtrahieren normalisiert kurz	N	7B	4	R1,D2(X2,B2)
SER	Subtrahieren normalisiert kurz	N	3B	2	R1,R2
SH	Subtrahieren Halbwort	N	4B	4	R1,D2(X2,B2)
SL	Subtrahieren ohne Vorzeichen	N	5F	4	R1,D2(X2,B2)
SLA	Verschieben links	N	8B	4	R1,D2(B2)
SLDA	Verschieben links doppelt	N	8F	4	R1,D2(B2)
SLDL	Verschieben links doppelt logisch	N	8D	4	R1,D2(B2)
SLL	Verschieben links logisch	N	89	4	R1,D2(B2)
SLR	Subtrahieren ohne Überlauf	N	1F	2	R1,R2
SP	Subtrahieren dezimal	N	FB	6	D1(L1,B1),D2(L2,B2)
** SPKA	Set PSW Key from Address	X	B20A	4	D2(B2)
SPM	Setzen Programmaske	N	04	2	R1
SR	Subtrahieren	N	1B	2	R1,R2
SRA	Verschieben rechts	N	8A	4	R1,D2(B2)
SRDA	Verschieben rechts doppelt	N	8E	4	R1,D2(B2)
SRDL	Verschieben rechts doppelt logisch	N	8C	4	R1,D2(B2)
SRL	Verschieben rechts logisch	N	88	4	R1,D2(B2)
SRP	Verschieben und Runden dezimal	N	F0	6	D1(L1,B1),D2(B2),I3
* SSCH	Start Subchannel	X	B233	4	D2(B2)
* SSK	Setzen Speicherschlüssel	N	08	2	R1,R2
* SSM	Speichern aus Schattenspeicher	N	DA	6	D1(L,B1),D2(B2)
* SSP	Speichern aus Zwischenspeicher	N	D0	6	D1(L,B1),D2(B2)
ST	Speichern	N	50	4	R1,D2(X2,B2)
STAM	Store Access Multiple	E	9B	4	R1,R3,D2(B2)
STC	Speichern Zeichen	N	42	4	R1,D2(X2,B2)
STCK	Speichern Uhrzeit	N	B2	4	D1(B1)
STCM	Speichern Zeichen mit Maske	N	BE	4	R1,M3,D2(B2)
* STCPS	Store Channel Path Status	X	B23A	4	D2(B2)
* STCRW	Store Channel Report Word	X	B239	4	D2(B2)
STD	Speichern lang	N	60	4	R1,D2(X2,B2)
STE	Speichern kurz	N	70	4	R1,D2(X2,B2)
STH	Speichern Halbwort	N	40	4	R1,D2(X2,B2)
STM	Speichern mehrfach	N	90	4	R1,R3,D2(B2)
* STSCH	Store Subchannel	X	B234	4	D2(B2)
SU	Subtrahieren nicht normalisiert kurz	N	7F	4	R1,D2(X2,B2)
SUR	Subtrahieren nicht normalisiert kurz	N	3F	2	R1,R2
SVC	Aufrufen Organisationsprogramm	N	0A	2	I

Mnem. Code	Befehlsname	NXS XS ESA	Masch. Code	Länge	Operandenformat
SW	Subtrahieren nicht normalisiert lang	N	6F	4	R1,D2(X2,B2)
SWR	Subtrahieren nicht normalisiert lang	N	2F	2	R1,R2
SXR	Subtrahieren normalisiert mit erweiterter Länge	N	37	2	R1,R2
TAR	Test Access Register	E	B24C	4	R1,R2
* TDV	Prüfen Gerät	N	9D	4	D1(B1)
TM	Testen mit Maske	N	91	4	D1(B1),I2
* TPI	Test Pending Interruption	X	B236	4	D2(B2)
TR	Umsetzen Code	N	DC	6	D1(L,B1),D2(B2)
* TRACE	Trace	X	99	4	R1,R3,D2(B2)
TRT	Umsetzen und Testen	N	DD	6	D1(L,B1),D2(B2)
TS	Testen und Setzen	N	93	4	D1(B1)
* TSCH	Test Subchannel	X	B235	4	D2(B2)
UNPK	Entpacken	N	F3	6	D1(L1,B1),D2(L2,B2)
* WRD	Schreiben direkt	N	84	4	D1(B1),I2
X	Ausschließendes ODER	N	57	4	R1,D2(X2,B2)
XC	Ausschließendes ODER	N	D7	6	D1(L,B1),D2(B2)
XI	Ausschließendes ODER	N	97	4	D1(B1),I2
XR	Ausschließendes ODER	N	17	2	R1,R2
ZAP	Löschen und Addieren dezimal	N	F8	6	D1(L1,B1),D2(L2,B2)

* Privilegierte Befehle

** Semiprivilegierte Befehle

11.3 Assembler-Restriktionen

Der ASSEMBH kann keine beliebig großen Programme übersetzen. Es gibt Einschränkungen bezüglich der Größe der Programme, der Zahl der Namen, und ähnlichem, die beachtet werden müssen, damit die Übersetzung korrekt abläuft. Diese ASSEMBH-spezifischen Einschränkungen sind im folgenden aufgeführt. Zusätzlich zu diesen theoretischen Maximalwerten ist die Größe von Programmen, die der ASSEMBH übersetzen kann, abhängig von der Speicherkapazität und der Komplexität der Programme und des Assemblers.

- **Anzahl der Zeilen im Quellprogramm**

Im Quellprogramm gilt jede einzelne Instruktion und jeder Kommentar als ein Statement und erhält im Übersetzungsprotokoll eine Statementnummer. Wird eine Instruktion über mehrere Listingzeilen fortgesetzt, dann hat trotzdem die gesamte Instruktion nur eine Statementnummer.

Ein Quellprogramm darf nach der Auflösung der Makros und dem Einfügen der COPY-Elemente maximal 2^{31}-1 Statements enthalten.

- **Anzahl der Namen und Literale**

Ein Quellprogramm darf maximal 2^{31}-1 Namen und Literale enthalten, d.h., die Symboltabelle darf maximal so viele Einträge haben.
Wurden in einem Quellprogramm mehr Namen und Literale verwendet, gibt der ASSEMBH eine Meldung aus und die Übersetzung wird abgebrochen.

- **Länge von Namen**

Namen dürfen eine maximale Länge von 64 Zeichen haben. Externe Namen in Moduln im OM-Format (Objektmodul-Format), die vom Binder TSOSLNK bearbeitet werden, sind auf 8 Zeichen beschränkt.
Externe Namen in Moduln, die mit der Option COMPILER-ACT(,MODULE-FORMAT=LLM) erstellt wurden (siehe ASSEMBH, Benutzerhandbuch [1]), sind auf 32 Zeichen erweitert und werden vom Binder BINDER bearbeitet.

- **Klammerstufen in arithmetischen Ausdrücken**

Das Ergebnis des Ausdrucks

```
(Anzahl Elemente) + (Anzahl Operatoren) + (Klammerstufen) + (Verwaltungseinträ-
ge)
```

darf den Grenzwert nicht überschreiten, der durch die internen Tabellen des Assemblers vorgegeben ist.

- **Anzahl der Makrodefinitionen**

Aus den Bibliotheken, die für eine Übersetzung zugewiesen wurden und dem Quelltext können maximal 2^{31}-1 Makrodefinitionen verwendet werden.

- **Schachtelungstiefe von Makros und COPY-Elementen**

Die maximal mögliche Schachtelungstiefe von Makros ist 255.

Für COPY-Elemente ist die Schachtelungstiefe auf 5 voreingestellt. Sie kann mit Hilfe einer SDF-Option auf maximal 255 erhöht werden (siehe ASSEMBH, Benutzerhandbuch [1]).

- **Länge der Operanden von Makroaufrufen**

Ein Operand eines Makroaufrufs darf maximal 1020 Zeichen lang sein.

Bei Kennwortoperanden wird die Zeichenzahl nach dem Gleichheitszeichen gezählt. Enthält ein Operand variable Parameter, gilt diese Länge nach dem Einsetzen der aktuellen Werte. Ist der Operand eine Operandenunterliste, dann werden alle Kommas und Klammern als Zeichen mitgezählt.

- **Variable Parameter**

Pro Makrodefiniton dürfen maximal 2^{15}-1 variable Parameter verwendet werden.

- **Länge der Eingabesätze**

Beim Einlesen aus einer Datei ist festgelegt, was der Assembler als Quelltext interpretieren soll. Dieser Quelltext darf maximal 255 Zeichen lang sein.

Die Voreinstellung für den zu interpretierenden Quelltext sind die Spalten 1, 71, 72 und 16. Mit Hilfe der SDF-Option FROM-COLUMN, TO-COLUMN
kann die Einstellung für die Anfangsspalte bis auf 70, die für die Endspalte auf maximal 255 erhöht werden. Mit der
ICTL-Anweisung kann die Anfangsspalte auf 40 und die Endspalte auf maximal 80 eingestellt werden.

- **XREF-Listing**

Im XREF-Listing können maximal 2^{31}-1 Referenzen stehen.

Wird diese Zahl überschritten, dann erfolgt eine Fehlermeldung und Abbruch der Übersetzung.

- **ESID-Nummern**

Der ASSEMBH kann maximal 2^{15}-1 ESID-Nummern vergeben.

Eine ESID-Nummer bekommt:
– jeder Programmabschnitt (Typ SD)
– jeder gemeinsame Hilfsabschnitt (Typ CM)
– jeder Pseudoabschnitt (Typ DS)
– jeder externe Pseudoabschnitt (Typ XD, XR)
– jeder XDSEC-Name (Typ XD, XR)
– jeder EXTRN-Name (Typ ER)
– jeder WXTRN-Name (Typ WX)
– jede V-Konstante (Typ VC)
– jedes Pseudoregister (Typ DX)

- **Assembler-/Makroanweisungen**

Anzahl der LTORG-Anweisungen: beliebig
maximaler Wiederholungsfaktor bei DC und DS: 2^{24}-1
maximaler Längenfaktor bei DC und DS: abhängig vom Konstantentyp
maximale Längenangabe bei EQU: 2^{24}-1
maximale Zahl von Folgesymbolen pro Makro: 2^{15}-1
maximale Dimension bei indizierten SET-Parametern: 2^{31}-1
maximaler Wertebereich von arithmetischen Ausdrücken: -2^{31} bis 2^{31}-1
Anzahl der Fortsetzungszeilen einer Instruktion: 9
Fortsetzungszeilen im alternativen Anweisungsformat: beliebig

Zusätzliche Restriktionen bei der Strukturierten Programmierung

- **Vordefinierte Namen**

Alle Namen, die mit "@" oder mit "R@" beginnen, sind für ASSEMBH reserviert und dürfen nicht anderweitig verwendet werden.

Mehrzweckregister können mit R0 bis R15 angesprochen werden, Gleitpunktregister mit FA bis FD.

Registerverwendung

Nur die Register 5 bis 9 können uneingeschränkt verwendet werden. Alle anderen sind durch die folgenden Registerkonventionen vorbesetzt.

Register	Verwendung
1	Adresse der Parameterliste bei der Parameterübergabe
1 bis 4	Parameterwerte, bzw. -adressen bei der Parameterübergabe in OPTIMAL-Form
10	Basisadreßregister für Prozeduren der Typen M, E und I
11	Arbeitsregister. Bei C-Programmen Zeiger auf Static-Bereiche
12	Anfangsadresse des Program Manager
13	Basisadreßregister für den STACK
14	Prozedur-Rücksprungsadresse
15	Prozedur-Anfangsadresse, Standard-Basisadreßregister für Prozeduren der Typen B und L

- **Durch ASSEMBH generierte Namen**

Es sind maximal 9000 interne Namen möglich, die vom ASSEMBH generiert werden können. Bei einem Überlauf der internen Namenszählung gibt der ASSEMBH eine MNOTE mit Abbruchgewicht aus.

- **Verschachtelung von Strukturblöcken**

Die Verschachtelung von Strukturblöcken wird in einem SETC-Parameter abgelegt. Die Verschachtelung ist durch die Länge dieses Parameters begrenzt. Wird er zu lang, erfolgt eine Fehlermeldung.

- **Modulgröße**

Die Größe eines Objektmoduls ist begrenzt durch die Adressierbarkeit über ein Basisadreßregister.

11.4 Pseudoregister, Beispiele

```
PSEUDOREGISTER BEISPIEL1/ 1. TEIL                                                14:27:36  1994-03-08   PAGE 0003

LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             1             TITLE 'PSEUDOREGISTER BEISPIEL1/ 1. TEIL'
                                             2     *
                                             3     * - IN DIESEM BEISPIEL WIRD BEREITS ZUM ZEITPUNKT DES PROGRAMMIERENS
                                             4     *   DIE LAENGE DES PSEUDOREGISTERVEKTORS BESTIMMT UND FESTGELEGT.
                                             5     * - DIE ADRESSIERUNG DER EINZELNEN PSEUDOREGISTER ERFOLGT UEBER
                                             6     *   Q-KONSTANTEN
                                             7     *
                                             8             PRINT NOGEN
 000000                                      9     PSEUDO1  START
                                            10             EXTRN PSEUDO2
 0000F4                                     11             ENTRY PSRVEKT
 000000 05 A0                               12             BALR  10,0
 000002                 00000002            13             USING *,10
                                            14     *
 000002 41 10 A0F2      000000F4            15             LA    1,PSRVEKT
 000006 58 20 A0E6      000000E8            16             L     2,QPSREG1               R2 <- DISTANZ DES PSREG. 1
 00000A 1A 21                               17             AR    2,1                     ZUM ANFANG DES PSREG.VEKTORS
 00000C D2 09 2000A142           00000144   18             MVC   0(10,2),=C'WUNDERLICH'  WIRD INHALT DES PSREG 1
 000012 58 20 A0EE      000000F0            19             L     2,QPSREG3               R2 <- DISTANZ DES PSREG 3
 000016 1A 21                               20             AR    2,1                     ZUM ANFANG DES PSREG.VEKTORS
 000018 D2 4F 2000A094           00000096   21             MVC   0(80,2),TEXT            TEXT WIRD INHALT DES PSREG 3
 00001E 58 E0 A13E      00000140            22             L     14,=A(PSEUDO2)
 000022 05 FE                               23             BALR  15,14
 000024                                     24             WROUT NACHR1,FEHLER
                                            39   2               *,@DCEO      952    900503   53531004            00066200
                                            42   1               *,WROUT      004    890217   53121058
                                            43     *
 000032                                     44             TERM
                                            47   2               *,VERSION 010                                    00001300
 000046                                     59     FEHLER   TERM  DUMP=Y
                                            62   2               *,VERSION 010                                    00001300
                                            74     *
 00005C                                     75     NACHR1   DS    0F
 00005C 0038000040                          76             DC    X'0038000040'           X'38' = NACHRICHTENLAENGE
 000061 5C5C5C5C5C5C5C5C                    77             DC    C'**************PROGRAMMENDE BEISPIEL 1***************'
 000096 C5C9D540D4C5D5E2                    78     TEXT    DC    C'EIN MENSCH KANNS MANCHMAL NICHT VERSTEHN,'
 0000BF E3D9C9C6C6E340C5                    79             DC    C'TRIFFT EIN WAS ER VORAUSGESEHN         '
                                            80     *
 000000                                     81     PSREG1   DXD   CL10                   DEFINITIONEN DER PSREG 1,
 000000                                     82     PSREG2   DXD   CL60                                    2 UND
 000000                                     83     PSREG3   DXD   20F                                     3
                                            84     *
 0000E8 00000000                            85     QPSREG1  DC    Q(PSREG1)              DEFINITINEN DER ZUR ADRESSIERUNG
 0000EC 00000000                            86     QPSREG2  DC    Q(PSREG2)              DER EINZELNEN PSREG VERWENDETEN
 0000F0 00000000                            87     QPSREG3  DC    Q(PSREG3)              Q-KONSTANTEN
                                            88     *
 0000F4                                     89     PSRVEKT  DS    CL(L'PSREG1+L'PSREG2+L'PSREG3)     PSEUDOREGISTERVEKTOR
                                            90     *
                                            91             END
 000140 00000000                            92                   =A(PSEUDO2)
 000144 E6E4D5C4C5D9D3C9                    93                   =C'WUNDERLICH'


FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-03-08 AT 14:07:24
```

```
BEISPIEL 1/ 2.TEIL                                                                  14:27:37  1994-03-08   PAGE 0003


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             1     *
                                             2               TITLE 'BEISPIEL 1/ 2.TEIL'
                                             3     *
                                             4               PRINT NOGEN
 000000                                      5     PSEUDO2   START
 000000 05 B0                                6               BALR  11,0
 000002                00000002              7               USING *,11
                                             8               EXTRN PSRVEKT
                                             9     *
 000002 58 10 B016     00000018             10               L     1,=A(PSRVEKT)
 000006 58 20 B012     00000014             11               L     2,QPSREG2               R2 <- DISTANZ DES PSREG 2 ZUM
 00000A 1A 21                               12               AR    2,1                     ANFANG DES PSREG.VEKTORS
 00000C D2 13 2000B01A          0000001C    13               MVC   0(20,2),=C'(E.ROTH          )'
 000012 07 FF                               14               BR    15
                                            15     *
 000000                                     16     PSREG2    DXD   CL22
 000014 00000000                            17     QPSREG2   DC    Q(PSREG2)
                                            18     *
                                            19               END
 000018 00000000                            20                     =A(PSRVEKT)
 00001C 4DC54BD9D6E3C840                    21                     =C'(E.ROTH          )'


FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-03-08 AT 14:07:24
```

```
PSEUDOREGISTER BEISPIEL 2/ 1. TEIL                                    14:28:39  1994-03-08   PAGE 0003


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             1            TITLE 'PSEUDOREGISTER BEISPIEL 2/ 1. TEIL'
                                             2     *
                                             3     * - IN DIESEM BEISPIEL WIRD DER PSEUDOREGISTERVEKTOR ZUR PROGRAMM-
                                             4     *   LAUFZEIT DURCH ANFORDERUNG VON SPEICHERSEITEN FESTGELEGT. DER
                                             5     *   PROGRAMMIERER MUSS SICH NICHT UM DAS ERMITTELN DER LAENGE DES
                                             6     *   PSEUDOREGISTERVEKTORS KUEMMERN, DA DIESE VOM BINDER IN DAS FELD
                                             7     *   'PRVLEN' (SIEHE INITIALISIERUNGSROUTINE) EINGETRAGEN WIRD. DIE
                                             8     *   AUSWERTUN GERFOLGT ZUR PROGRAMMLAUFZEIT.
                                             9     * - DIE ADRESSIERUNG DER EINZELNEN PSEUDOREGISTER WIRD UEBER EIN
                                            10     *   'PSEUDOREGISTERVEKTOR-BASISREGISTER' BEWERKSTELLIGT.
                                            11     *
                                            12            PRINT NOGEN
000000                                      13     PSEUDO1  START
                                            14            EXTRN PRVINIT
000000 05 A0                               15            BALR  10,0
000002                 00000002             16            USING *,10
000002                                      17            USING *PRV,8                  FESTLEGEN DES BASISREGISTERS FUR
                                            18     *                                    ZUGRIFFE AUF DEN PSREG.VEKTOR
000002 58 E0 A0E6      000000E8             19            L     14,=A(PRVINIT)
000006 05 CE                                20            BALR  12,14
000008 18 81                                21            LR    8,1                     R8 <- A(1.BEREITGESTELLTE SEITE)
                                            22     *
00000A D2 23 8000A0EA 00000000 000000EC     23            MVC   PSREG1,=C'EIN MENSCH WIRD MUEDE SEINER FRAGEN:'
000010 D2 03 8000A07B 00000000 0000007D     24            MVC   PSREG2,TEXT2            UEBERTRAGEN VON DATEN
000016 D2 20 8000A09F 00000000 000000A1     25            MVC   PSREG3,TEXT3            AUF DIE
00001C D2 07 8000A0C0 00000000 000000C2     26            MVC   PSREG4,TEXT4            PSEUDOREGISTER
000022                                      27            WROUT NACHR,FEHLER
                                            42   2              *,@DCEO      952    900503   53531004           00066200
                                            45   1              *,WROUT      004    890217   53121058
                                            46     *
000032                                      47     FEHLER   TERM
                                            50   2              *,VERSION 010                                    00001300
                                            62     *
000048                                      63     NACHR    DS    0F
000048 0035000040                           64            DC    X'0035000040'
00004D 5C5C5C5C5C5C5C5C                     65            DC    C'************PROGRAMMENDE BEISPIEL 2***********'
00007D D5C9C540D2C1D5D5                     66     TEXT2    DC    C'NIE KANN DIE WELT IHM ANTWORT SAGEN.'
0000A1 C4D6C3C840C7C5D9                     67     TEXT3    DC    C'DOCH GERN GIBT AUSKUNFT ALLE WELT'
0000C2 C1E4C640C6D9C1C7                     68     TEXT4    DC    C'AUF FRAGEN, DIE ER NIE GESTELLT. '
                                            69     *
000000                                      70     PSREG1   DXD   CL36
000000                                      71     PSREG2   DXD   9F
000000                                      72     PSREG3   DXD   XL33
000000                                      73     PSREG4   DXD   4D
                                            74     *
000000                                      75            END   PSEUDO1
0000E8 00000000                             76                  =A(PRVINIT)
0000EC C5C9D540D4C5D5E2                     77                  =C'EIN MENSCH WIRD MUEDE SEINER FRAGEN:'


FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-03-08 AT 14:07:24
```

```
PSEUDOREGISTER BEISPIEL 2/ INITIALISIERUNG                                    14:28:40  1994-03-08   PAGE 0003


 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                              1     *
                                              2               TITLE 'PSEUDOREGISTER BEISPIEL 2/ INITIALISIERUNG'
                                              3               PRINT NOGEN
                                              4     *
 000000                                       5     PRVINIT   START
 000000 05 B0                                 6               BALR  11,0
 000002                 00000002              7               USING *,11
                                              8     *
 000002 17 22                                 9               XR    2,2                     ERMITTELN D. ENTSPR. DER LAENGE
 000004 58 30 B036      00000038             10               L     3,PRVLEN                DES PSREG.VEKTORS NOTWENDIGEN
 000008 5D 20 B03E      00000040             11               D     2,=F'4096'              ANZAHL 4K-SEITEN UND
 00000C 5A 30 B042      00000044             12               A     3,=F'1'
                                             13     *
 000010                                      14               REQM  (3)                     BEREITSTELLEN DERSELBEN
                                             17   2                 *,VERSION 500                                    00001300
 00001E 12 FF                                24               LTR   15,15
 000020 07 8C                                25               BRZ   12
                                             26     *
 000022                                      27               TERM  DUMP=Y
                                             30   2                 *,VERSION 010                                    00001300
                                             42     *
 000038                                      43               DS    0F
 000038 00000000                             44     PRVLEN    CXD                           HIER HINTERLEGT BINDER DIE LAENGE
                                             45     *                                       DES PSREG.VEKTORS
 000000                                      46               END   PRVINIT
 000040 00001000                             47                     =F'4096'
 000044 00000001                             48                     =F'1'


FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NOTE
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-03-08 AT 14:07:24
```

11.5 Parameterübergabe bei der strukturierten Programmierung, Beispiel

```
ASSEMBH LISTING                                                                    14:29:44  1994-03-08   PAGE 0003


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             1      DEMOPARA START
                                             2              PRINT NOGEN
                                             3     * MATRIXBEARBEITUNG EINER ZEILENWEISE
                                             4     * GESPEICHERTEN MATRIX VON WORTEN
000000                                       5     DEMOPARA @ENTR TYP=M,MAXPRM=2,                                          *
                                             5                    FUNCT='DEMO MOEGLICHKEITEN PARAMETERUEBERGABE/NAHME'
                                            89   3               *,VERSION 010                                        00001300
                                           110     * DIMENSION MATRIX ERRECHNEN UND SPEICHER ANFORDERN
000030 48 B0 A13E     00000140             111              LH    R11,ANZZEIL
000034 4C B0 A140     00000142             112              MH    R11,ANZSPAL
000038 40 B0 A142     00000144             113              STH   R11,DIMMATR
00003C 1A BB                               114              AR    R11,R11
00003E 1A BB                               115              AR    R11,R11
000040                                     116              @DATA CLASS=A,BASE=R11,LENGTH=(R11)
                                           127     *
000050 41 60 A144     00000146             128              LA    ARZEILE,HZEILE
000054 41 70 A146     00000148             129              LA    ARSPALTE,HSPALTE
                                           130     * MATRIX ELEMENTWEISE ABARBEITEN
000058 41 80 0001                          131              LA    RZEILE,1
                                           132              @THRU (RZEILE),ANZZEIL
00005C                                     136              @DO
000064 40 80 A144     00000146             140              STH   RZEILE,HZEILE
000068 41 90 0001                          141              LA    RSPALTE,1
                                           142              @THRU (RSPALTE),ANZSPAL
00006C                                     146              @DO
000074 40 90 A146     00000148             150              STH   RSPALTE,HSPALTE
                                           151              PRINT GEN
                                           152              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0004


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                           153     * DYNAMISCHE PARAMETERUEBERGABE (OPTIMAL)
                                           154     *
                                           155     * UEBERGABE ADRESSE VON ZEILEN- UND SPALTENNUMMER ALS REGISTERINHALT
                                           156     * REGISTER VERWEISEN AUF PARAMETERWERTE
                                           157               @PASS NAME=DO1,PASS=OPT,PLIST=((ARZEILE),(ARSPALTE))
                                           158   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           159   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           160   1           @@PPP P,R@STACK,96,2,OPT,((ARZEILE),(ARSPALTE))
000078 18 16                               161   2           LR    R1,ARZEILE
00007A 18 27                               162   2           LR    R2,ARSPALTE
00007C 58 F0 A15A     0000015C             163   1           L     R@PASS,=A(DO1)
                                           164   1         ##BALR  R@EXIT,R@PASS
000080 05 EF                               165   2           BALR  R@EXIT,R@PASS
                                           166     * UEBERGABE ADRESSE VON ZEILEN- UND SPALTENNUMMER ALS REGISTERINHALT
                                           167     * REGISTER VERWEISEN AUF PARAMETERWERTE
                                           168               @PASS NAME=DO2,PASS=OPT,PLIST=(HZEILE,HSPALTE)
                                           169   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           170   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           171   1           @@PPP P,R@STACK,96,2,OPT,(HZEILE,HSPALTE)
000082 41 10 A144     00000146             172   2           LA    R1,HZEILE
000086 41 20 A146     00000148             173   2           LA    R2,HSPALTE
00008A 58 F0 A15E     00000160             174   1           L     R@PASS,=A(DO2)
                                           175   1         ##BALR  R@EXIT,R@PASS
00008E 05 EF                               176   2           BALR  R@EXIT,R@PASS
                                           177     * UEBERGABE ADRESSE VON ZEILEN- UND SPALTENNUMMER ALS REGISTERINHALT
                                           178     * REGISTER VERWEISEN AUF PARAMETERWERTE
                                           179               @PASS NAME=DO3,PASS=OPT,PLIST=(HZEILE,HSPALTE)
                                           180   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           181   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           182   1           @@PPP P,R@STACK,96,2,OPT,(HZEILE,HSPALTE)
000090 41 10 A144     00000146             183   2           LA    R1,HZEILE
000094 41 20 A146     00000148             184   2           LA    R2,HSPALTE
000098 58 F0 A162     00000164             185   1           L     R@PASS,=A(DO3)
                                           186   1         ##BALR  R@EXIT,R@PASS
00009C 05 EF                               187   2           BALR  R@EXIT,R@PASS
                                           188     * UEBERGABE ZEILEN- UND SPALTENNUMMER ALS REGISTERINHALT
                                           189     * REGISTER ENTHALTEN PARAMETERWERTE
                                           190               @PASS NAME=DO4,PASS=OPT,PLIST=((RZEILE),(RSPALTE))
                                           191   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           192   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           193   1           @@PPP P,R@STACK,96,2,OPT,((RZEILE),(RSPALTE))
00009E 18 18                               194   2           LR    R1,RZEILE
0000A0 18 29                               195   2           LR    R2,RSPALTE
0000A2 58 F0 A166     00000168             196   1           L     R@PASS,=A(DO4)
                                           197   1         ##BALR  R@EXIT,R@PASS
0000A6 05 EF                               198   2           BALR  R@EXIT,R@PASS
                                           199               EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0005

 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                            200     * DYNAMISCHE PARAMETERUEBERGABE (STANDARD)
                                            201     *
                                            202     * UEBERGABE ADRESSE DYNAMISCHE PARAMETERADRESSENLISTE
                                            203     * REGISTER 1 VERWEIST AUF PARAMETERADRESSENLISTE
                                            204     * LISTEINTRAEGE VERWEISEN AUF PARAMETERWERTE
                                            205               @PASS NAME=DS1,PASS=STA,PLIST=((ARZEILE),(ARSPALTE))
                                            206   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                            207   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                            208   1           @@PPP P,R@STACK,96,2,STA,((ARZEILE),(ARSPALTE))
0000A8 50 60 D060                           209   2           ST    ARZEILE,96(0,R@STACK)
0000AC 50 70 D064                           210   2           ST    ARSPALTE,100(0,R@STACK)
0000B0 41 10 D060                           211   2           LA    R@PAR,96(0,R@STACK)
0000B4 96 80 D064                           212   2           OI    100(R@STACK),X'80'
0000B8 58 F0 A16A     0000016C              213   1           L     R@PASS,=A(DS1)
                                            214   1         ##BALR  R@EXIT,R@PASS
0000BC 05 EF                                215   2           BALR  R@EXIT,R@PASS
                                            216     * UEBERGABE ADRESSE DYNAMISCHE PARAMETERADRESSENLISTE
                                            217     * REGISTER 1 VERWEIST AUF PARAMETERADRESSENLISTE
                                            218     * LISTEINTRAEGE VERWEISEN AUF PARAMETERWERTE
                                            219               @PASS NAME=DS2,PASS=STA,PLIST=(HZEILE,HSPALTE)
                                            220   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                            221   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                            222   1           @@PPP P,R@STACK,96,2,STA,(HZEILE,HSPALTE)
0000BE 41 E0 A144     00000146              223   2           LA    R@EXIT,HZEILE
0000C2 50 E0 D060                           224   2           ST    R@EXIT,96(0,R@STACK)
0000C6 41 E0 A146     00000148              225   2           LA    R@EXIT,HSPALTE
0000CA 50 E0 D064                           226   2           ST    R@EXIT,100(0,R@STACK)
0000CE 41 10 D060                           227   2           LA    R@PAR,96(0,R@STACK)
0000D2 96 80 D064                           228   2           OI    100(R@STACK),X'80'
0000D6 58 F0 A16E     00000170              229   1           L     R@PASS,=A(DS2)
                                            230   1         ##BALR  R@EXIT,R@PASS
0000DA 05 EF                                231   2           BALR  R@EXIT,R@PASS
                                            232     * UEBERGABE ADRESSE DYNAMISCHE PARAMETERADRESSENLISTE
                                            233     * REGISTER 1 VERWEIST AUF PARAMETERADRESSENLISTE
                                            234     * LISTEINTRAEGE VERWEISEN AUF PARAMETERWERTE
                                            235               @PASS NAME=DS3,PASS=STA,PLIST=(HZEILE,HSPALTE)
                                            236   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                            237   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                            238   1           @@PPP P,R@STACK,96,2,STA,(HZEILE,HSPALTE)
0000DC 41 E0 A144     00000146              239   2           LA    R@EXIT,HZEILE
0000E0 50 E0 D060                           240   2           ST    R@EXIT,96(0,R@STACK)
0000E4 41 E0 A146     00000148              241   2           LA    R@EXIT,HSPALTE
0000E8 50 E0 D064                           242   2           ST    R@EXIT,100(0,R@STACK)
0000EC 41 10 D060                           243   2           LA    R@PAR,96(0,R@STACK)
0000F0 96 80 D064                           244   2           OI    100(R@STACK),X'80'
0000F4 58 F0 A172     00000174              245   1           L     R@PASS,=A(DS3)
                                            246   1         ##BALR  R@EXIT,R@PASS
0000F8 05 EF                                247   2           BALR  R@EXIT,R@PASS
                                            248               EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0006


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                           249     * STATISCHE PARAMETERUEBERGABE
                                           250     *
                                           251     * UEBERGABE ADRESSE STATISCHE PARAMETERADRESSENLISTE
                                           252     * REGISTER 1 VERWEIST AUF PARAMETERADRESSENLISTE
                                           253     * LISTEINTRAEGE VERWEISEN AUF PARAMETERWERTE
                                           254               @PASS NAME=S1,PAR=STAPLIS
                                           255   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           256   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
0000FA 41 10 A12E     00000130             257   1           LA    R@PAR,STAPLIS
0000FE 58 F0 A176     00000178             258   1           L     R@PASS,=A(S1)
                                           259   1         ##BALR  R@EXIT,R@PASS
000102 05 EF                               260   2           BALR  R@EXIT,R@PASS
                                           261     * UEBERGABE ADRESSE STATISCHE PARAMETERADRESSENLISTE
                                           262     * REGISTER 1 VERWEIST AUF PARAMETERADRESSENLISTE
                                           263     * LISTEINTRAEGE VERWEISEN AUF PARAMETERWERTE
                                           264               @PASS NAME=S2,PAR=STAPLIS
                                           265   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           266   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
000104 41 10 A12E     00000130             267   1           LA    R@PAR,STAPLIS
000108 58 F0 A17A     0000017C             268   1           L     R@PASS,=A(S2)
                                           269   1         ##BALR  R@EXIT,R@PASS
00010C 05 EF                               270   2           BALR  R@EXIT,R@PASS
                                           271     * UEBERGABE ADRESSE STATISCHE PARAMETERADRESSENLISTE
                                           272     * REGISTER 1 VERWEIST AUF PARAMETERADRESSENLISTE
                                           273     * LISTEINTRAEGE VERWEISEN AUF PARAMETERWERTE
                                           274               @PASS NAME=S3,PAR=STAPLIS
                                           275   1           @@OUV ,,O,U,V,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
                                           276   2           @@SYN K,@PASS,(@,T,E,D,K,G,Y,A,X,Z,B)
00010E 41 10 A12E     00000130             277   1           LA    R@PAR,STAPLIS
000112 58 F0 A17E     00000180             278   1           L     R@PASS,=A(S3)
                                           279   1         ##BALR  R@EXIT,R@PASS
000116 05 EF                               280   2           BALR  R@EXIT,R@PASS
                                           281               PRINT NOGEN
000118                                     282               @BEND
000120                                     289               @BEND
000128                                     296               @EXIT
                                           303               PRINT GEN
                                           304     * STATISCHE PARAMETERADRESSENLISTE
                                           305     STAPLIS  @PAR  PLIST=(ZNA,SNA),VLIST=(HZEILE,HSPALTE)
                                           306   1           @@SYN ,@PAR,,LE,1
000130                                     307   1 STAPLIS  DS    0F
000130 00000146                            308   1 ZNA      DC    A(HZEILE)
000134 80000148                            309   1 SNA      DC    A(HSPALTE+X'80000000')
                                           310               EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0007


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             311             PRINT NOGEN
                      00000138               312     VAR     EQU   *
000138 00000146                              313     ADRZEIL DC    A(HZEILE)
00013C 00000148                              314     ADRSPAL DC    A(HSPALTE)
000140                                       315     ANZZEIL DS    H
000142                                       316     ANZSPAL DS    H
000144                                       317     DIMMATR DS    H
000146                                       318     HZEILE  DS    H
000148                                       319     HSPALTE DS    H
                      00000006               320     ARZEILE EQU   R6
                      00000007               321     ARSPALTE EQU  R7
                      00000008               322     RZEILE  EQU   R8
                      00000009               323     RSPALTE EQU   R9
000150                                       324             @END
                      00000009               343     Z       EQU   R9
                      00000008               344     S       EQU   R8
                      00000007               345     P       EQU   R7
                                             346             PRINT GEN
                                             347     *
                                             348     * PARAMETERUEBERNAHME
                                             349             EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0008


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             350     * UEBERNAHME PARAMETERADRESSEN IN REGISTERN R1,R2
                                             351     DO1      @ENTR TYP=I,TITLE=NO
                                             352   1          @@SYN ,@ENTR,,EQ,0
000188                                       353   1 DO1      DS    0D
000188                00000000               354   1          USING @SAV,R@STACK
000188 90 EC D00C     0000000C               355   1          STM   R14,R12,@SAVR14
00018C 18 AF                                 356   1          LR    R@BASE,R@PASS
00018E                00000188               357   1          USING DO1,R@BASE
                                             358   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(96),CL8'DO1')
00018E 58 F0 A038     000001C0               359   2          L     R@PASS,=V($NUCENTR)
000192                                       360   2          CNOP  2,4
                                             361   2        ##BALR  R@EXIT,R@PASS
000192 05 EF                                 362   3          BALR  R@EXIT,R@PASS
000194 00000060                              363   2          DC    A(96)
000198 C4D6F14040404040                      364   2          DC    CL8'DO1'
                                             365             PRINT NOGEN
0001A0                                       366             @DATA CLASS=S,BASE=R12,INIT=VAR
                                             371     * ERRECHNEN MATRIXPOSITION
0001A4 48 70 1000                            372             LH    P,0(0,R1)     WERT ERSTER PARAMETER
0001A8 06 70                                 373             BCTR  P,0
0001AA 4C 70 C00A     00000142               374             MH    P,ANZSPAL
0001AE 4A 70 2000                            375             AH    P,0(0,R2)     WERT ZWEITER PARAMETER
0001B2 06 70                                 376             BCTR  P,0
0001B4 1A 77                                 377             AR    P,P
0001B6 1A 77                                 378             AR    P,P
                                             379     * FUNKTION AUSFUEHREN
0001B8                                       380             @EXIT
0001C0                                       388             @END
                                             396             PRINT GEN
                                             397             EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0009


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                           398     * UEBERNAHME PARAMETERADRESSEN AUS REGISTERN IN ADRESSLISTE
                                           399     DO2      @ENTR TYP=I,TITLE=NO,LOCAL=PARLI1,PASS=OPT,PLIST=(ADRZN,ADRSN)
                                           400   1          @@SYN ,@ENTR,,EQ,0
0001D0                                     401   1 DO2      DS    0D
0001D0                 00000000            402   1          USING @SAV,R@STACK
0001D0 90 EC D00C      0000000C            403   1          STM   R14,R12,@SAVR14
0001D4 18 AF                               404   1          LR    R@BASE,R@PASS
0001D6                 000001D0            405   1          USING DO2,R@BASE
                                           406   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(LPARLI1),CL8'DO2')
0001D6 58 F0 A048      00000218            407   2          L     R@PASS,=V($NUCENTR)
0001DA                                     408   2          CNOP  2,4
                                           409   2        ##BALR  R@EXIT,R@PASS
0001DA 05 EF                               410   3          BALR  R@EXIT,R@PASS
0001DC 00000068                            411   2          DC    A(LPARLI1)
0001E0 C4D6F24040404040                    412   2          DC    CL8'DO2'
                                           413   1          @DATA BASE=R@STACK,DSECT=PARLI1
0001E8                 00000000            414   2          USING PARLI1,R@STACK
                                           415   1          @@PPP E,R@PAR,0,0,OPT,(ADRZN,ADRSN)
0001E8 50 10 D064      00000064            416   2          ST    R1,ADRZN
0001EC 50 20 D060      00000060            417   2          ST    R2,ADRSN
                                           418              PRINT NOGEN
0001F0                                     419              @DATA CLASS=S,BASE=R12,INIT=VAR
                                           424     * ERRECHNEN MATRIXPOSITION
0001F4 58 20 D064      00000064            425              L     R2,ADRZN   ADRESSE ERSTER PARAMETER
0001F8 48 70 2000                          426              LH    P,0(0,R2)     WERT ERSTER PARAMETER
0001FC 06 70                               427              BCTR  P,0
0001FE 4C 70 C00A      00000142            428              MH    P,ANZSPAL
000202 58 20 D060      00000060            429              L     R2,ADRSN   ADRESSE ZWEITER PARAMETER
000206 4A 70 2000                          430              AH    P,0(0,R2)     WERT ZWEITER PARAMETER
00020A 06 70                               431              BCTR  P,0
00020C 1A 77                               432              AR    P,P
00020E 1A 77                               433              AR    P,P
                                           434     * FUNKTION AUSFUEHREN
000210                                     435              @EXIT
000218                                     443              @END
                                           451              PRINT GEN
                                           452     PARLI1   @PAR  D=YES,LEND=YES,PLIST=(ADRSN,ADRZN)
                                           453   1          @@SYN ,@PAR,,LE,1
000000                                     454   1 PARLI1   DSECT
000000                 00000060            455   1          ORG   *+96
000060                                     456   1 ADRSN    DS    A
000064                                     457   1 ADRZN    DS    A
                       00000068            458   1 LPARLI1  EQU   *-PARLI1
000228                                     459   1 DEMOPARA CSECT
                                           460              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0010


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             461     * UEBERNAHME PARAMETERADRESSEN AUS REGISTERN IN REGISTER
                                             462     DO3      @ENTR TYP=I,TITLE=NO,PASS=OPT,PLIST=((Z),(S))
                                             463   1          @@SYN ,@ENTR,,EQ,0
000228                                       464   1 DO3      DS    0D
000228                 00000000              465   1          USING @SAV,R@STACK
000228 90 EC D00C      0000000C              466   1          STM   R14,R12,@SAVR14
00022C 18 AF                                 467   1          LR    R@BASE,R@PASS
00022E                 00000228              468   1          USING DO3,R@BASE
                                             469   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(96),CL8'DO3')
00022E 58 F0 A040      00000268              470   2          L     R@PASS,=V($NUCENTR)
000232                                       471   2          CNOP  2,4
                                             472   2        ##BALR  R@EXIT,R@PASS
000232 05 EF                                 473   3          BALR  R@EXIT,R@PASS
000234 00000060                              474   2          DC    A(96)
000238 C4D6F34040404040                      475   2          DC    CL8'DO3'
                                             476   1          @@PPP E,R@PAR,0,0,OPT,((Z),(S))
000240 18 91                                 477   2          LR    Z,R1
000242 18 82                                 478   2          LR    S,R2
                                             479              PRINT NOGEN
000244                                       480              @DATA CLASS=S,BASE=R12,INIT=VAR
                                             485     * ERRECHNEN MATRIXPOSITION
000248 48 70 9000                            486              LH    P,0(0,Z)      WERT ERSTER PARAMETER
00024C 06 70                                 487              BCTR  P,0
00024E 4C 70 C00A      00000142              488              MH    P,ANZSPAL
000252 4A 70 8000                            489              AH    P,0(0,S)      WERT ZWEITER PARAMETER
000256 06 70                                 490              BCTR  P,0
000258 1A 77                                 491              AR    P,P
00025A 1A 77                                 492              AR    P,P
                                             493     * FUNKTION AUSFUEHREN
00025C                                       494              @EXIT
000268                                       502              @END
                                             510              PRINT GEN
                                             511              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0011


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             512     * UEBERNAHME PARAMETERWERTE AUS REGISTERN IN REGISTER
                                             513     DO4      @ENTR TYP=I,TITLE=NO,PASS=OPT,PLIST=((Z),(S))
                                             514   1          @@SYN ,@ENTR,,EQ,0
000278                                       515   1 DO4      DS    0D
000278                 00000000              516   1          USING @SAV,R@STACK
000278 90 EC D00C      0000000C              517   1          STM   R14,R12,@SAVR14
00027C 18 AF                                 518   1          LR    R@BASE,R@PASS
00027E                 00000278              519   1          USING DO4,R@BASE
                                             520   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(96),CL8'DO4')
00027E 58 F0 A038      000002B0              521   2          L     R@PASS,=V($NUCENTR)
000282                                       522   2          CNOP  2,4
                                             523   2        ##BALR  R@EXIT,R@PASS
000282 05 EF                                 524   3          BALR  R@EXIT,R@PASS
000284 00000060                              525   2          DC    A(96)
000288 C4D6F44040404040                      526   2          DC    CL8'DO4'
                                             527   1          @@PPP E,R@PAR,0,0,OPT,((Z),(S))
000290 18 91                                 528   2          LR    Z,R1
000292 18 82                                 529   2          LR    S,R2
                                             530              PRINT NOGEN
000294                                       531              @DATA CLASS=S,BASE=R12,INIT=VAR
                                             536     * ERRECHNEN MATRIXPOSITION
000298 18 79                                 537              LR    P,Z           WERT ERSTER PARAMETER
00029A 06 70                                 538              BCTR  P,0
00029C 4C 70 C00A      00000142              539              MH    P,ANZSPAL
0002A0 1A 78                                 540              AR    P,S           WERT ZWEITER PARAMETER
0002A2 06 70                                 541              BCTR  P,0
0002A4 1A 77                                 542              AR    P,P
0002A6 1A 77                                 543              AR    P,P
                                             544     * FUNKTION AUSFUEHREN
0002A8                                       545              @EXIT
0002B0                                       553              @END
                                             561              PRINT GEN
                                             562              EJECT
```

```
 ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0012


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             563     * UEBERNAHME ADRESSLISTE VON PARAMETERN ADRESSIERT DURCH R1
                                             564     DS1      @ENTR TYP=I,TITLE=NO
                                             565   1          @@SYN ,@ENTR,,EQ,0
0002C0                                       566   1 DS1      DS    0D
0002C0                 00000000              567   1          USING @SAV,R@STACK
0002C0 90 EC D00C      0000000C              568   1          STM   R14,R12,@SAVR14
0002C4 18 AF                                 569   1          LR    R@BASE,R@PASS
0002C6                 000002C0              570   1          USING DS1,R@BASE
                                             571   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(96),CL8'DS1')
0002C6 58 F0 A040      00000300              572   2          L     R@PASS,=V($NUCENTR)
0002CA                                       573   2          CNOP  2,4
                                             574   2        ##BALR  R@EXIT,R@PASS
0002CA 05 EF                                 575   3          BALR  R@EXIT,R@PASS
0002CC 00000060                              576   2          DC    A(96)
0002D0 C4E2F14040404040                      577   2          DC    CL8'DS1'
                                             578              PRINT NOGEN
0002D8                                       579              @DATA CLASS=S,BASE=R12,INIT=VAR
                                             584     * ERRECHNEN MATRIXPOSITION
0002DC 58 20 1000                            585              L     R2,0(0,R1) ADRESSE ERSTER PARAMETER
0002E0 48 70 2000                            586              LH    P,0(0,R2)     WERT ERSTER PARAMETER
0002E4 06 70                                 587              BCTR  P,0
0002E6 4C 70 C00A      00000142              588              MH    P,ANZSPAL
0002EA 58 20 1004                            589              L     R2,4(0,R1) ADRESSE ZWEITER PARAMETER
0002EE 4A 70 2000                            590              AH    P,0(0,R2)     WERT ZWEITER PARAMETER
0002F2 06 70                                 591              BCTR  P,0
0002F4 1A 77                                 592              AR    P,P
0002F6 1A 77                                 593              AR    P,P
                                             594     * FUNKTION AUSFUEHREN
0002F8                                       595              @EXIT
000300                                       603              @END
                                             611              PRINT GEN
                                             612              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0013

LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                           613     * UEBERNAHME ADRESSLISTE VON PARAMETERN IN ADRESSLISTE
                                           614     DS2      @ENTR TYP=I,TITLE=NO,LOCAL=PARLI2,PASS=STA,PLIST=(ZNADR,SNADR)
                                           615   1          @@SYN ,@ENTR,,EQ,0
000310                                     616   1 DS2      DS    0D
000310                 00000000            617   1          USING @SAV,R@STACK
000310 90 EC D00C      0000000C            618   1          STM   R14,R12,@SAVR14
000314 18 AF                               619   1          LR    R@BASE,R@PASS
000316                 00000310            620   1          USING DS2,R@BASE
                                           621   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(LPARLI2),CL8'DS2')
000316 58 F0 A050      00000360            622   2          L     R@PASS,=V($NUCENTR)
00031A                                     623   2          CNOP  2,4
                                           624   2        ##BALR  R@EXIT,R@PASS
00031A 05 EF                               625   3          BALR  R@EXIT,R@PASS
00031C 00000068                            626   2          DC    A(LPARLI2)
000320 C4E2F24040404040                    627   2          DC    CL8'DS2'
                                           628   1          @DATA BASE=R@STACK,DSECT=PARLI2
000328                 00000000            629   2          USING PARLI2,R@STACK
                                           630   1          @@PPP E,R@PAR,0,0,STA,(ZNADR,SNADR)
000328 D2 03 D0641000 00000064             631   2          MVC   ZNADR(4),0(R@PAR)
00032E D2 03 D0601004 00000060             632   2          MVC   SNADR(4),4(R@PAR)
                                           633              PRINT NOGEN
000334                                     634              @DATA CLASS=S,BASE=R12,INIT=VAR
                                           639     * ERRECHNEN MATRIXPOSITION
000338 58 20 D064      00000064            640              L     R2,ZNADR   ADRESSE ERSTER PARAMETER
00033C 48 70 2000                          641              LH    P,0(0,R2)     WERT ERSTER PARAMETER
000340 06 70                               642              BCTR  P,0
000342 4C 70 C00A      00000142            643              MH    P,ANZSPAL
000346 58 20 D060      00000060            644              L     R2,SNADR   ADRESSE ZWEITER PARAMETER
00034A 4A 70 2000                          645              AH    P,0(0,R2)     WERT ZWEITER PARAMETER
00034E 06 70                               646              BCTR  P,0
000350 1A 77                               647              AR    P,P
000352 1A 77                               648              AR    P,P
                                           649     * FUNKTION AUSFUEHREN
000354                                     650              @EXIT
000360                                     658              @END
                                           666              PRINT GEN
                                           667     PARLI2   @PAR  D=YES,LEND=YES,PLIST=(SNADR,ZNADR)
                                           668   1          @@SYN ,@PAR,,LE,1
000000                                     669   1 PARLI2   DSECT
000000                 00000060            670   1          ORG   *+96
000060                                     671   1 SNADR    DS    A
000064                                     672   1 ZNADR    DS    A
                       00000068            673   1 LPARLI2  EQU   *-PARLI2
000370                                     674   1 DEMOPARA CSECT
                                           675              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0014


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                           676     * UEBERNAHME ADRESSEN AUS PARAMETERADRESSENLISTE IN REGISTER
                                           677     DS3      @ENTR TYP=I,TITLE=NO,PASS=STA,PLIST=((R2),(R1))
                                           678   1          @@SYN ,@ENTR,,EQ,0
000370                                     679   1 DS3      DS    0D
000370                 00000000            680   1          USING @SAV,R@STACK
000370 90 EC D00C      0000000C            681   1          STM   R14,R12,@SAVR14
000374 18 AF                               682   1          LR    R@BASE,R@PASS
000376                 00000370            683   1          USING DS3,R@BASE
                                           684   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(96),CL8'DS3')
000376 58 F0 A040      000003B0            685   2          L     R@PASS,=V($NUCENTR)
00037A                                     686   2          CNOP  2,4
                                           687   2        ##BALR  R@EXIT,R@PASS
00037A 05 EF                               688   3          BALR  R@EXIT,R@PASS
00037C 00000060                            689   2          DC    A(96)
000380 C4E2F34040404040                    690   2          DC    CL8'DS3'
                                           691   1          @@PPP E,R@PAR,0,0,STA,((R2),(R1))
000388 58 20 1000                          692   2          L     R2,0(0,R@PAR)
00038C 58 10 1004                          693   2          L     R1,4(0,R@PAR)
                                           694              PRINT NOGEN
000390                                     695              @DATA CLASS=S,BASE=R12,INIT=VAR
                                           700     * ERRECHNEN MATRIXPOSITION
000394 48 70 2000                          701              LH    P,0(0,R2)     WERT ERSTER PARAMETER
000398 06 70                               702              BCTR  P,0
00039A 4C 70 C00A      00000142            703              MH    P,ANZSPAL
00039E 4A 70 1000                          704              AH    P,0(0,R1)     WERT ZWEITER PARAMETER
0003A2 06 70                               705              BCTR  P,0
0003A4 1A 77                               706              AR    P,P
0003A6 1A 77                               707              AR    P,P
                                           708     * FUNKTION AUSFUEHREN
0003A8                                     709              @EXIT
0003B0                                     717              @END
                                           725              PRINT GEN
                                           726              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0015


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                             727     * UEBERNAHME ADRESSLISTE VON PARAMETERN ADRESSIERT DURCH R1
                                             728     S1       @ENTR TYP=I,TITLE=NO
                                             729   1          @@SYN ,@ENTR,,EQ,0
0003C0                                       730   1 S1       DS    0D
0003C0                 00000000              731   1          USING @SAV,R@STACK
0003C0 90 EC D00C      0000000C              732   1          STM   R14,R12,@SAVR14
0003C4 18 AF                                 733   1          LR    R@BASE,R@PASS
0003C6                 000003C0              734   1          USING S1,R@BASE
                                             735   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(96),CL8'S1')
0003C6 58 F0 A040      00000400              736   2          L     R@PASS,=V($NUCENTR)
0003CA                                       737   2          CNOP  2,4
                                             738   2        ##BALR  R@EXIT,R@PASS
0003CA 05 EF                                 739   3          BALR  R@EXIT,R@PASS
0003CC 00000060                              740   2          DC    A(96)
0003D0 E2F1404040404040                      741   2          DC    CL8'S1'
                                             742              PRINT NOGEN
0003D8                                       743              @DATA CLASS=S,BASE=R12,INIT=VAR
                                             748     * ERRECHNEN MATRIXPOSITION
0003DC 58 20 1000                            749              L     R2,0(0,R1) ADRESSE ERSTER PARAMETER
0003E0 48 70 2000                            750              LH    P,0(0,R2)     WERT ERSTER PARAMETER
0003E4 06 70                                 751              BCTR  P,0
0003E6 4C 70 C00A      00000142              752              MH    P,ANZSPAL
0003EA 58 20 1004                            753              L     R2,4(0,R1) ADRESSE ZWEITER PARAMETER
0003EE 4A 70 2000                            754              AH    P,0(0,R2)     WERT ZWEITER PARAMETER
0003F2 06 70                                 755              BCTR  P,0
0003F4 1A 77                                 756              AR    P,P
0003F6 1A 77                                 757              AR    P,P
                                             758     * FUNKTION AUSFUEHREN
0003F8                                       759              @EXIT
000400                                       767              @END
                                             775              PRINT GEN
                                             776              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0016


LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                           777     * UEBERNAHME ADRESSLISTE VON PARAMETERN IN ADRESSLISTE
                                           778     S2       @ENTR TYP=I,TITLE=NO,LOCAL=PARLI3,PASS=STA,PLIST=(AVZN,AVSN)
                                           779   1          @@SYN ,@ENTR,,EQ,0
000410                                     780   1 S2       DS    0D
000410                 00000000            781   1          USING @SAV,R@STACK
000410 90 EC D00C      0000000C            782   1          STM   R14,R12,@SAVR14
000414 18 AF                               783   1          LR    R@BASE,R@PASS
000416                 00000410            784   1          USING S2,R@BASE
                                           785   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(LPARLI3),CL8'S2')
000416 58 F0 A050      00000460            786   2          L     R@PASS,=V($NUCENTR)
00041A                                     787   2          CNOP  2,4
                                           788   2        ##BALR  R@EXIT,R@PASS
00041A 05 EF                               789   3          BALR  R@EXIT,R@PASS
00041C 00000068                            790   2          DC    A(LPARLI3)
000420 E2F2404040404040                    791   2          DC    CL8'S2'
                                           792   1          @DATA BASE=R@STACK,DSECT=PARLI3
000428                 00000000            793   2          USING PARLI3,R@STACK
                                           794   1          @@PPP E,R@PAR,0,0,STA,(AVZN,AVSN)
000428 D2 03 D0641000 00000064             795   2          MVC   AVZN(4),0(R@PAR)
00042E D2 03 D0601004 00000060             796   2          MVC   AVSN(4),4(R@PAR)
                                           797              PRINT NOGEN
000434                                     798              @DATA CLASS=S,BASE=R12,INIT=VAR
                                           803     * ERRECHNEN MATRIXPOSITION
000438 58 20 D064      00000064            804              L     R2,AVZN    ADRESSE ERSTER PARAMETER
00043C 48 70 2000                          805              LH    P,0(0,R2)     WERT ERSTER PARAMETER
000440 06 70                               806              BCTR  P,0
000442 4C 70 C00A      00000142            807              MH    P,ANZSPAL
000446 58 20 D060      00000060            808              L     R2,AVSN    ADRESSE ZWEITER PARAMETER
00044A 4A 70 2000                          809              AH    P,0(0,R2)     WERT ZWEITER PARAMETER
00044E 06 70                               810              BCTR  P,0
000450 1A 77                               811              AR    P,P
000452 1A 77                               812              AR    P,P
                                           813     * FUNKTION AUSFUEHREN
000454                                     814              @EXIT
000460                                     822              @END
                                           830              PRINT GEN
                                           831     PARLI3   @PAR  D=YES,LEND=YES,PLIST=(AVSN,AVZN)
                                           832   1          @@SYN ,@PAR,,LE,1
000000                                     833   1 PARLI3   DSECT
000000                 00000060            834   1          ORG   *+96
000060                                     835   1 AVSN     DS    A
000064                                     836   1 AVZN     DS    A
                       00000068            837   1 LPARLI3  EQU   *-PARLI3
000470                                     838   1 DEMOPARA CSECT
                                           839              EJECT
```

```
ASSEMBH LISTING                                                    14:29:44  1994-03-08   PAGE 0017


 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
                                            840     * UEBERNAHME ADRESSEN AUS PARAMETERADRESSENLISTE IN REGISTER
                                            841     S3       @ENTR TYP=I,TITLE=NO,PASS=STA,PLIST=((R2),(R1))
                                            842   1          @@SYN ,@ENTR,,EQ,0
 000470                                     843   1 S3       DS    0D
 000470                 00000000            844   1          USING @SAV,R@STACK
 000470 90 EC D00C      0000000C            845   1          STM   R14,R12,@SAVR14
 000474 18 AF                               846   1          LR    R@BASE,R@PASS
 000476                 00000470            847   1          USING S3,R@BASE
                                            848   1          @PASS EXTNAME=$NUCENTR,CNOP=(0,4),DC=(A(96),CL8'S3')
 000476 58 F0 A040      000004B0            849   2          L     R@PASS,=V($NUCENTR)
 00047A                                     850   2          CNOP  2,4
                                            851   2        ##BALR  R@EXIT,R@PASS
 00047A 05 EF                               852   3          BALR  R@EXIT,R@PASS
 00047C 00000060                            853   2          DC    A(96)
 000480 E2F3404040404040                    854   2          DC    CL8'S3'
                                            855   1          @@PPP E,R@PAR,0,0,STA,((R2),(R1))
 000488 58 20 1000                          856   2          L     R2,0(0,R@PAR)
 00048C 58 10 1004                          857   2          L     R1,4(0,R@PAR)
                                            858              PRINT NOGEN
 000490                                     859              @DATA CLASS=S,BASE=R12,INIT=VAR
                                            864     * ERRECHNEN MATRIXPOSITION
 000494 48 70 2000                          865              LH    P,0(0,R2)     WERT ERSTER PARAMETER
 000498 06 70                               866              BCTR  P,0
 00049A 4C 70 C00A      00000142            867              MH    P,ANZSPAL
 00049E 4A 70 1000                          868              AH    P,0(0,R1)     WERT ZWEITER PARAMETER
 0004A2 06 70                               869              BCTR  P,0
 0004A4 1A 77                               870              AR    P,P
 0004A6 1A 77                               871              AR    P,P
                                            872     * FUNKTION AUSFUEHREN
 0004A8                                     873              @EXIT
 0004B0                                     881              @END
                                            889              END


FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH     V 1.2A00     ON 1994-03-08 AT 14:07:24
THIS LISTING WAS GENERATED BY THE LISTING GENERATOR V 1.2A00.
```

11.6 Unterschiede von ASSEMBH V1.2A und ASSEMB V30.0A

In den folgenden Tabellen sind die Funktionen und Sprachelemente aufgeführt, bei denen sich Unterschiede von ASSEMBH V1.2A und ASSEMB V30.0A ergeben. Nicht aufgeführte Funktionen bleiben gleich.

x Funktion wird unterstützt
– Funktion wird nicht unterstützt

	ASSEMBH V1.2A	ASSEMB V30.0A
SDF-Benutzerschnittstelle	x	-
/PARAM-Kommando	-	x
Prozeßschaltersteuerung bei DUET-Programmierung	-	x
Prozeßschaltersteuerung für die MCALL-Option	-	x
COMOPT-Schnittstelle	x	x
*COMOPT ADIAG=	x (nicht ASSEMBH-BC)	x
COPYMAC	-	x
ERRFIL	x	x
ISD	x (nicht ASSEMBH-BC) Erstellung von LSD-Information	Erstellung von ISD-Sätzen
MCALL	-	x
MDIAG	-	x
OUTPUT	-	x
PROCOM	-	x
SAVLST	x (nicht ASSEMBH-BC)	x
SEQ	x	x
SOURCE = +	- max. 255 Zeichen	max. 8 Zeichen
SYSPARM	- bei Verwendung nicht unterstützter Optionen gibt der ASSEMBH eine Meldung aus	x
UPD		
Ausgabe von Objektcode		
im OM-Format	x	x
im LLM-Format	x	-

	ASSEMBH V1.2A	ASSEMB V30.0A
AID-Schnittstelle	x (nicht ASSEMBH-BC)	-
Ablage der AID-Konstante im Objektmodul	x Nach dem ersten Programmabschnitt wird bei der Option TEST-SUPPORT=AID eine 8-Bytes lange Konsistenzkonstante im Objekt abgelegt.	-
IDA-Schnittstelle	-	x
ASSDIAG / ADIAG	x (nicht ASSEMBH-BC)	x
Kommandos COMOPT, OPEN	-	x
Expliziter Start	-	x
CDT-Anweisungen	x	-
Kommando CONTINUE-CDT	x	-
Zahl der möglichen MAKRO- und COPY-Bibliotheken	je 100	insgesamt 5
ISLU	-	x
HALSTEAD-Metrik	-	x
Unterstützung der Groß-/ Kleinschreibung	x	-
Maximale Zahl der ESID-Nummern	$2^{31}-1$	2500
Maximale Zahl der Programmabschnitte	$2^{15}-1$	512
Vergabe der Statementnummer	Alle Statements, außer Makroanweisungen bei der Makrogenerierung und Makrokommentare, erhalten eine Statementnummer, wenn sie fehlerhaft sind oder MTRAC angegeben ist. Eine Fortsetzungszeile erhält die Nummer der Anfangszeile des Statements.	Die Statementnummer wird abhängig von PRINT-Steuerung, Fehlermeldungen und MTRAC-Steuerung vergeben. Jedes gedruckte Statement erhält eine eigene Nummer. In Makrodefinitionen erhält jede Fortsetzungszeile eine eigene Nummer.

Assemblerbefehle, Assembleranweisungen, Kommentare

	ASSEMBH V1.2A	ASSEMB V30.0A
Länge von Namen Externe Namen Unterstrich in Namen	64 Zeichen 8/32 Zeichen x	8 Zeichen 8 Zeichen -
Namen als Elemente in Ausdrücken	müssen nicht vorher definiert sein (Ausnahme: im Operand der ORG-Anweisung)	müssen vor ihrer Verwendung in Anweisungen definiert sein
Arithmetische Ausdrücke	beliebige Schachtelung der Klammern müssen immer in Klammern stehen	maximal 6 Klammerstufen Ausdrücke, die nur Ziffern enthalten, können ohne Klammer stehen (Fehler des Assemblers)
Fortsetzungszeilen	beliebig viele	3
Assemblerbefehle EXST LBF LWI POP PUSH STBF STWI ESA-Befehle	 - - - - - - - x	 x x x x x x x -
ISEQ-Anweisung	- die Anweisung wird wie ein Kommentar behandelt.	x
COPY-Anweisung	Schachtelungstiefe maximal 255	Schachtelungstiefe maximal 5
DC-Anweisung	Wiederholungsfaktor bis $2^{24}-1$	Wiederholungsfaktor bis $2^{16}-1$

	ASSEMBH V1.2A	ASSEMB V30.0A
EJECT-Anweisung	wird vor dem Vorschub protokolliert bei fehlerhaften Operanden wird eine MNOTE ausgegeben.	wird nicht protokolliert, fehlerhafte Operanden werden ignoriert (Fehler des Assembler).
EQU-Anweisung	negative Werte sind möglich	nur positive Werte
LTORG-Anweisung	beliebige Anzahl im Quellprogramm	maximal 255
Fehlende LTORG-Anweisung	der Literalbereich wird nach der END-Anweisung protokolliert	der Literalbereich wird vor der END-Anweisung protokolliert
Fehlerhafte Literale	werden nicht in den Literalbereich aufgenommen	werden auch in den Literalbereich aufgenommen
Literalablage im Literalpool und Literal-XREF	Semantisch gleiche, aber syntaktisch verschiedene Literale werden in Literalpool und -XREF je Literaldefinition abgelegt. Beispiel: Die Literale '=A(B)' und '=AL4(B)' werden beide abgelegt. Syntaktisch gleiche Literale werden intern in Literalpool und -XREF nur einmal abgelegt. Durch die unterschiedliche Literalablage sind Abweichungen im Objekt möglich, was unter Umständen zu nicht mehr passenden REPs führen kann.	Semantisch gleiche, aber syntaktisch verschiedene Literale werden in Literalpool und -XREF nur einmal abgelegt. Für syntaktisch gleiche Literale gilt das zum ASSEMBH geschriebene.
MNOTE-Anweisung	fehlt das Leerzeichen zwischen Operand und Bemerkung, erfolgt eine Meldung	fehlt das Leerzeichen, so wird nach dem Apostroph alles als Bemerkung interpretiert (Fehler des Assembler)

	ASSEMBH V1.2A	ASSEMB V30.0A
OPSYN-Anweisung Bearbeitung von Modellanweisungen in Makros	auch innerhalb von Makros für Makroanweisungen, sowie REPRO- und COPY-Anweisung gilt das OPSYN bei Definition des Makro; für alle übrigen Instruktionen gilt das OPSYN, das direkt davor liegt	nur ausserhalb von Makros für Bibliotheks-Makros gilt das letzte OPSYN in der Quelle, für Makros in der Quelle gilt das OPSYN vor der Definition.
PRINT-Anweisung Operanden BASE CLOSED/OPEN DECK/NODECK NUM/NONUM NOBF/DBF REF/NOREF xON/xOFF	 für jedes Basisregister wird zusätzlich der Kommentar aus der USING-Anweisung ausgegeben - - - - - -	 es wird nur der adressierbare Bereich ausgegeben x x x x x x
PUNCH-Anweisung REPRO-Anweisung	Werden bei der Modulausgabe im LLM-Format nicht mehr unterstützt	kein LLM-Format
SPACE-Anweisung	wird vor dem Vorschub protokolliert bei fehlerhaften Operanden wird eine MNOTE ausgegeben	wird nicht protokolliert fehlerhafte Operanden werden ignoriert (Fehler des Assembler)
TITLE-Anweisung	wird vor dem Vorschub protokolliert der Namenseintrag wird in einen PLAM-Objektmodul nicht ausgegben	wird nach dem Vorschub protokolliert immer Ausgabe des Namenseintrags in den Objektmodul

	ASSEMBH V1.2A	ASSEMB V30.0A
Register 0	Ausgabe einer Warnung, wenn ein Speicheroperand mit Basis- und Indexregister 0 adressiert wird	Ausgabe einer Warnung, wenn der Assembler für den 2. Operanden einer SS-Instruktion bei fehlendem Basisregister den Wert 0 einsetzt.
XDSEC-Anweisung ohne Operand, bzw. mit fehlerhaftem Operand	es wird R angenommen, falls nicht bereits ein gleichnamiger externer Pseu. definiert ist	Ausgabe einer Warnung
Kommentarzeilen mit .* am Anfang im Assembler-Quellprogramm	werden wie in einer Makrodefinition behandelt und nicht ausgedruckt	sind im Assembler-Quellprogramm nicht erlaubt

Makrosprachelemente

	ASSEMBH V1.2A	ASSEMB V30.0A
Makrodefinitionen im Quellprogramm	die Definition muß vor dem 1. Aufruf des Makro durchlaufen werden	die Definition kann an beliebiger Stelle im Quellprogramm stehen
Umdefinieren von Assemblerinstruktionen (in Makrodefinitionen im Quellprogramm)	der mnemotechnische Operationscode von Assemblerinstruktionen kann als Makroname verwendet werden; der jeweilige Makro ersetzt die entsprechende Instruktion	umdefinieren von mnemotechnischem Operationscode ist nur über die OPSYN-Anweisung möglich
Generierte mnemotechnische Operationscodes von Assembleranweisungen	die Anweisungen ICTL, COPY, MNOTE und REPRO dürfen nicht generiert werden	die Anweisungen ICTL, START, COM, MNOTE, REPRO und COPY dürfen nicht generiert werden
conditional assembly	Makrodefinitionen im Quellprogramm werden nur eingelesen, wenn sie durchlaufen werden; Bibliotheksmakros werden nur eingelesen, wenn ihr Aufruf durchlaufen wurde	Makrodefinitionen im Quellprogramm werden immer eingelesen; Bibliotheksmakros werden immer eingelesen, wenn ihr Aufruf im Quellprogramm steht
Innere Makrodefinitionen	Makrodefinitionen dürfen auch innerhalb einer anderen stehen	Makrodefinitionen dürfen nicht innerhalb von anderen stehen

	ASSEMBH V1.2A	ASSEMB V30.0A
Makrodefinitionen mit gleichem Namen im Quellprogramm oder in einer Makrodefinition	Es gilt die zuletzt eingelesene Definition bis zur nächsten.	Es gilt die erste Definition. Weitere werden als Fehler behandelt.
Kommentarzeilen mit .* am Anfang	sind auch vor der Musteranweisung möglich dürfen nicht generiert werden	sind nur nach der Musteranweisung möglich dürfen generiert werden
Bezugnahmen auf Merkmale	führen auch bei Bezugnahme auf noch nicht definierte Namen zu gültigen Merkmalen	bei noch nicht definierten Namen sind die Merkmale immer undefiniert.
Bezugnahme auf das Definitionsmerkmal (D')	x	-
Bezugnahme auf das Anzahlmerkmal (N')	ergibt bei SET-Parametern die aktuelle Dimension	ist für SET-Parameter nicht erlaubt
Generierte Folgesymbole	werden immer erkannt und bearbeitet	werden nur erkannt, wenn sie in einer GSEQ-Anweisung definiert sind
Folgesymbole im Namenseintrag von generierten Instruktionen	werden nicht protokolliert	werden protokolliert
Mehrfach definierte Folgesymbole	Fehlermeldung	Fehlermeldung nur, wenn MTRAC eingeschaltet ist
Länge von Zeichenwerten und Teilzeichenfolgen	maximal 1020 Zeichen	maximal 255 Zeichen

	ASSEMBH V1.2A	ASSEMB V30.0A
Verkettung von Teilzeichenfolgen	Der Verkettungspunkt vor dem Wiederholungsfaktor der 2. Folge kann weggelassen werden.	Der Verkettungspunkt muß stehen.
Schreibweise von Zeichenwerten	Die Schreibweise C'...' ist nicht zulässig	C'...' ist zulässig Das C wird ignoriert (Fehler des Assembler)
Arithmetischer Makroausdruck	Zeichenwerte sind als Elemente nicht zulässig	Zeichenwerte werden konvertiert. Ist nicht möglich, erfolgt keine Meldung und es wird mit 0 weitergerechnet (Fehler des Assembler)

Makroaufruf und Musteranweisung

	ASSEMBH V1.2A	ASSEMB V30.0A
Makroaufruf und Musteranweisung	Stellungs- und Kennwortparameter können in gemischter Reihenfolge angegeben werden	Zuerst müssen alle Stellungsparameter kommen, dann alle Kennwortparameter
Innere Makroaufrufe	Innere Makroaufrufe werden immer aufgelöst, wenn sie durchlaufen werden	Bei Angabe der MCALL-Option werden innere Makroaufrufe nur aufgelöst, wenn sie in einer MCALL-Anweisung definiert sind
Adreßpegel des Makroaufrufs	Der aktuelle Adreßpegel wird nicht protokolliert	Der aktuelle Adreßpegel wird protokolliert
Generierte Makronamen	Generierte Makronamen werden immer erkannt und bearbeitet	Generierte Makronamen werden nur erkannt und aufgelöst, wenn sie in einer MCALL-Anweisung definiert sind
Länge der Operanden von Makroaufrufen	maximal 1020 Zeichen	maximal 127 Zeichen
Ausdrücke in Makroaufruf-Operanden	ist der Operand eines Makroaufrufs ein Ausdruck, so hat der entsprechende symb. Parameter die Merkmale einer C-Konstanten	ist der Operand eines Makroaufrufs ein Ausdruck,so hat der symb. Parameter die Merkmale des 1. Namen im Ausdruck.

Makroanweisungen

	ASSEMBH V1.2A	ASSEMB V30.0A
ACTR-Anweisung	der ACTR-Zähler ist vorbesetzt mit 4096	der ACTR-Zähler ist vorbesetzt mit 1200
AIF- und AGO-Anweisung	jeweils erweitertes Format möglich (siehe 7.2), jeweils alternatives Anweisungsformat möglich	- nur normales Format möglich
LCLx- und GBLx-Anweisung	jeweils alternatives Anweisungsformat möglich	nur normales Format möglich
OPSYN-Anweisung	auch innerhalb einer Makrodefinition möglich	nur außerhalb der Makrodefinition möglich
GSEQ-Anweisung MCALL-Anweisung	- - beide Anweisungen werden mit einer Warnung versehen, ansonsten ignoriert	x x
Protokollierung von sedezimalen selbstdefinierenden Werten	werden dezimal protokolliert	die Original-Instruktion wird protokolliert

Variable Parameter

	ASSEMBH V1.2A	ASSEMB V30.0A
Generierte Parameternamen	x	-
Implizite Definition von lokalen SET-Parametern	x	-
SETA-Parameter	einem SETA-Parameter kann ein SETC-Parameter zugewiesen werden, wenn er einen arithmetischen Wert hat	einem SETA-Parameter kann ein beliebiger Zeichenausdruck zugewiesen werden, der einen arithmetischen Wert hat
SETB-Parameter	einem SETB-Parameter kann ein SETC-Parameter zugewiesen werden, wenn sein Wert 0 oder 1 ist.	nicht zulässig
Variable Systemparameter &SYSECT &SYSLIST(n) mit n=0 &SYSPARM &SYSTSEC &SYSVERM/&SYSVERS	 wird auch mit Namen von COM und XDSEC belegt ergibt den Namenseintrag des Makroaufrufs maximal 255 Zeichen x werden rechts abgeschnitten, bzw. rechts mit Unterstrich '_' aufgefüllt	 wird nur durch START, CSECT und DSECT belegt nicht erlaubt maximal 8 Zeichen - werden rechts abschnitten, bzw. links mit 0 aufgefüllt füllt
Variable Systemparameter im Operationseintrag	der Operationseintrag kann mit variablen Systempar. generiert werden	variable Systempar. sind im Operationseintrag nicht erlaubt
Verkettung von variablen Parametern mit alphanum. Zeichen	bei Verwendung einer Verkettung in Assemblerbefehlen muß der Verkettungspunkt geschrieben werden	bei einer Verkettung in Assemblerbefehlen kann der Verkettungspunkt wegfallen (Fehler des Assembler)

Strukturierte Programmierung mit ASSEMBH

		ASSEMBH V1.2A	ASSEMB V30.0A/ COLUMBUS-ASSEMBLER V2.2F
STACK-Verwaltung		dynamische STACK-Verwaltung; Der logische Aufbau des STACK entspricht der PROSYS-Standard Linkage-Konvention	statische Ablage im Speicher
@ENTR-Makro			
Operanden	ENTRY=	x	-
	AMODE=	x	-
	RMODE=	x	-
	STACK=	x Größe des dynamisch erweiterbaren Initialstacks: Keine Angabe = eine Seite (4096 Bytes). STACK=n bedeutet: n Bytes Init.stack	x Minimalstack, dynamisch erweiterbar (keine STACK-Angabe) oder statischer, nicht erweiterbarer Stack von n Bytes (STACK=n).
	KL5SP=	- eine KL5SP-Angabe wird mit MNOTE 0 ignoriert.	x
	ILCS=YES	x	-
	STREQ	x	-
	STREL	x	-
	HPREQ	x	-
	HPREL	x	-
	SLTERM	x	-
	SCTERM	x	-
	EXTMIN	x	-
	ABKR	x	-
	PROCHK	x	-
	ERROR	x	-
	OTHEVT	x	-
Entry IASSIN (COLBIN)		ignoriert eine KL5SP-Angabe	unterstützt eine KL5SP-Angabe

	ASSEMBH V1.2A	ASSEMB V30.0A/ COLUMBUS-ASSEMBLER V2.2F
@CONDI-Makro @CONEN-Makro @EVTLC-Makro @EVTOE-Makro @ININ -Makro @SETJV-Makro @SETPM-Makro @STXDI-Makro @STXEN-Makro @STXIM-Makro	x x x x x x x x x x	- - - - - - - - - -
@END-Makro Operand DROP=	x zusätzlicher Operandenwert =(), alle USING-Register werden freigegeben	x

12 Handbuchergänzungen

Diese Kapitel aktualisiert das vorliegende Handbuch auf den Stand ASSEMBH V1.2D.

12.1 SPACE-Anweisung

SPACE Zeilenvorschub (S.123)

Beschreibung

nr modulo 100 gibt die Anzahl der Leerzeilen an, die nach der SPACE-Anweisung im Übersetzungsprotokoll erscheinen sollen.

Ist nr modulo 100 größer als die Zahl der auf dieser Seite noch verbleibenden Zeilen, dann bewirkt die SPACE-Anweisung einen Seitenvorschub.

12.2 Variable Systemparameter

Abschnitt 6.3, Zusätzlicher Systemparameter (S.197)

&SYSDATE_ISO4

Wert von &SYSDATE_ISO4: jjjj-mm-tt Zählermerkmal ist 10

Das Jahr wird 4-stellig ausgegeben, sonst alles wie bei &SYSDATE.

12.3 Prozeduren vom Typ S

Abschnitt 9.3.2.2 (S.288), ergänzen um:

Prozeduren vom Typ S

Prozeduren vom Typ S sind wie die Prozedurtypen B/L/D nicht an die Speicherverwaltung angeschlossen. Es wird kein neuer Sicherstellungsbereich bereitgestellt und es ist keine dynamische Parameterübergabe möglich. Im Gegensatz zu den Typen B/L/D werden aber die Registerstände der rufenden Prozedur in deren Sicherstellungsbereich gesichert.

Mit dem Parameter ENTRY= kann gesteuert werden, ob ein CSECT- oder eine ENTRY-Anweisung generiert wird.

Standardmäßig ist Register 15 als Basisadressregister zugewiesen. Im BASE-Operanden des @ENTR-Makro kann der Anwender ein anderes Register als Basisadressregister zuweisen. Dieses Register muss am Anfang der Prozedur mit dem richtigen Wert geladen werden (z.B. mit dem Assemblerbefehl "LR reg,R15").

Da die Prozeduren auf einer niedrigen Ebene liegen (kein Anschluss an das Laufzeitsystem), muss der Anwender selbst bei Bedarf die Register restaurieren, d.h. die Register werden bei Prozedurbeendigung nicht zurückgeladen.

Aus demselben Grund sollte der Aufruf weiterer Unterprozeduren mit @PASS vermieden werden, da wegen der fehlenden Savearea-Verkettung keine Rückverfolgung mehr möglich ist.

12.4 Prozedurverknüpfung und Parameterübergabe

Abschnitt 9.5, Prozedurverknüpfung und Parameterübergabe (S. 301)

Beim Aufruf von Prozeduren der Typen B, L, D und S wird kein Prozedur STACK bereitgestellt.

12.5 Statische Parameterübergabe

Abschnitt 9.5.1.1, Statische Parameterübergabe (S. 303)

Diese Form der Parameterübergabe ist erlaubt in Prozeduren vom Typ M, E, I, L, B und S.

12.6 Folgeprozesse behandeln

Abschnitt 9.6.4, Folgeprozesse behandeln (S. 322), vorletzter Absatz:

Danach ruft der Standard-Contingency-Handler die Benutzer Contingency-Routine nach ILCS-Konventionen auf. Die Benutzer-Contingency-Routine erhält die Informationen über den unterbrochenen Prozess durch R1.

R1 zeigt auf eine Parameterliste mit folgendem Inhalt:

1. Wort: Contingency-Mitteilung;
   entspricht dem CONMSG-Parameter im Makro @CONEN (S. 335) (siehe dazu im BS2000 Manual 'Makroaufrufe an den Ablaufteil', Kapitel 'Contingency-Prozesse', Abschnitt 'Informationsübergabe an Contingency-Prozesse')
2. Wort: Ereignis-Informationscode;
   (siehe Manual wie oben, gleicher Abschnitt, Tabelle 'Ereignis-Informationscode')
3. Wort: Pointer auf Post Code;
   (siehe Manual wie oben, gleicher Abschnitt)

12.7 Unterbrechungsereignisse behandeln

Abschnitt 9.6.5, Unterbrechungsereignisse behandeln (S. 323), 3. Absatz:

Von ILCS werden alle im BS2000 definierten STXIT-Ereignisse unterstützt. Die Ereignisse TERM und ABEND werden nur von ILCS intern behandelt jedoch nicht an eine vom Anwender über @STXEN angemeldete Routine weitergeleitet.

12.8 Vordefinierte Makros für die strukturierte Programmierung

Kapitel 10, Vordefinierte Makros für die strukturierte Programmierung

– @CONEN Contingency-Routine anmelden (S.335),
erster Absatz
Dieser Makroaufruf ist nur in Prozeduren mit ILCS=YES erlaubt.

  Er arbeitet nur dann ordnungsgemäß, wenn die Initialisierung mit ILCS richtig ausgeführt wurde.

– @CONEN Contingency-Routine anmelden (S.337), letzter Absatz

  Returncode in Register 15:

  0 Verarbeitung richtig abgeschlossen

  1 Register 13 hat einen falschen Wert

  2 ILCS ist nicht richtig initialisiert

  3 ungültiger Parameterwert bei CONLEV

  Alle anderen Returncodes signalisieren interne Fehler.

@ENTR (Typ=M/E/I), Parameter ILCS=YES (S. 354, 362) voller Anschluss an ILCS, erlaubt die Angabe der folgenden Parameter.

  @ENTR (Typ=M/E/I), Programmierhinweise (S. 357, 363)

  Das Abbruchkennzeichen in der Event-Handler-List wird benötigt, wenn der Standard-Event-Handler die Suche nach Ereignisbehandlungsroutinen abbrechen soll. Innerhalb der Prozedurkette existierende Nicht-ILCS- Prozeduren (für diese wird keine Event-Handler-List angelegt) werden von ILCS erkannt und übergangen.

  @ENTR (Typ=E/I), Parameter RETURNS=YES (S. 360)

  Bei ILCS=YES: Die Register 2 bis 14 werden zurückgeladen.

  @ENTR (Typ=B/L), Parameter TYP=B (S. 365)

  Gibt an, dass es sich um eine Prozedur handelt, die von einem beliebigen Modul aus aufgerufen werden kann (externe Prozedur) und die nicht an Speicherverwaltung und Registersicherung angeschlossen ist.

@ENTR mit neuem Prozedurtyp S (S. 367.1)

Format 5: Prozeduren vom Typ S

```
------------------------------------------------------------------
Name     | Operation | Operanden
---------|-----------|--------------------------------------------
[name]   | @ENTR     | TYP=S
|        | [,BASE=reg],
|        | [,ENTRY= {ENTRY|CSECT
|        |                 [,AMODE={24|31|ANY}
|        |                  ,RMODE={24|ANY}]}]
------------------------------------------------------------------
```

name Name der Prozedur

reg Mehrzweckregister; positiver absoluter Ausdruck, entweder

- dezimaler selbstdefinierender Wert oder
- vordefinierter Name eines Registers oder
- Name, dem ein entsprechender selbstdefinierender Wert zugewiesen wurde.

a) Beschreibung

Es handelt sich um eine Prozedur vom Typ S, ohne Anschluss an die Speicherverwaltung des Laufzeitsystems. Im Gegensatz zu den Prozeduren vom Typ B/L/D werden bei diesem Prozedurtyp die Registerstände der rufenden Prozedur gesichert.

BASE=reg Weist der Prozedur ein Register als Basisadressregister zu.

Ohne diese Angabe wird der Prozedur vom Assembler das Register 15 als Basisadressregister zugewiesen.

ENTRY= gibt an, welche Eingangsinstruktion für die Prozedur generiert werden soll.

Bei Angabe von ENTRY wird generiert:
```
name1  DS      0D
ENTRY          name1
```

Bei Angabe von CSECT wird generiert:
```
name1          CSECT
```

AMODE= ordnet der Prozedur einen Adressierungsmodus zu (siehe 4.2, AMODE-Anweisung)

RMODE= ordnet der Prozedur ein Ladeattribut zu
(siehe 4.2, RMODE-Anweisung)

Bei einer unzulässigen Kombination von AMODE und RMODE wird eine MNOTE generiert und für beide Werte AMODE 24 und RMODE 24 eingesetzt.

b) Programmierhinweise

1. Standardmäßig wird der Prozedur das Register 15 (=Einsprungpunkt) als Basisadressregister zugewiesen, falls mit BASE kein anderes Register vereinbart wurde.
2. Die Prozedur darf nicht die erste eines Moduls sein.
3. Der Prozedurtyp ist nur bei ILCS=NO zugelassen.

@ENTR (Typ=M), Parameter ENV=C (S. 354)

Die Einschränkung 'Nur bei ILCS=NO zugelassen' entfällt.

@ENTR (Typ=E/I), Parameter ENV=C (S. 361)

Die Angabe ist nur für Prozeduren vom Typ E zulässig.

@ENTR (Typ=E/I), Parameter ENTRY= (S. 361)

Die Angabe von @ENTR ENTRY= ist sowohl für Prozeduren vom Typ E als auch für Prozeduren vom Typ I zulässig.

Bei Angabe von @ENTR TYP=I zusammen mit den Parametern ENTRY=ENTRY und ENV=SPL... wird ein interner Einsprungpunkt generiert, dem ein Code-Basisregister zugewiesen ist.

@ENTR (Typ=E), Programmierhinweise (S. 363)

1. Das Abbruchkennzeichen ...
2. Mit @ENTR TYP=E und den Parametern ENV=C und ENTRY=ENTRY wird keine neue Prozedur, sondern lediglich ein Secondary Entry in der übergeordneten Prozedur generiert. Diese Definition darf abweichend von Prozeduren ohne Environment-Angabe nicht mit @END abgeschlossen werden.

@EXIT Format 1 (S. 372)

– Format 1: Rückkehr aus Prozeduren vom Typ M, B, L oder S

In Prozeduren vom Typ B,L oder S wird das Programm mit der Instruktion fortgesetzt, die in der rufenden Prozedur dem @PASS-Makro folgt.

@PASS Programmierhinweise (S. 389)

1. Eine mit EXTNAME=ext_name aufgerufene Prozedur muss mit @ENTR TYP=E/B/S vereinbart sein.

12.9 Pseudoregister, Beispiel

zu Abschnitt 11.4 Pseudoregister, Beispiel

In Beispiel 1/Teil 1 ist die Deklaration von PSRVEKT nicht korrekt.

Sie muss geändert werden in
PSRVEKT DS CL(L'PSREG1 + L'PSREG2 + 20 * L'PSREG3)

12.10 Unterschiede ASSEMBH V1.2A/ASSEMB V30.0A

zu Abschnitt 11.6 Unterschiede ASSEMBH V1.2A/ASSEMB V30.0A

Im Listing für DSECT wird kein Objektcode ausgedruckt, auch wenn ein DC oder Befehle in der DSECT stehen.

Literatur

[1] **ASSEMBH**
Benutzerhandbuch

[2] **AID (BS2000)**
Testen von ASSEMBH-Programmen
Benutzerhandbuch

[3] **BS2000**
Assemblerbefehle
Beschreibung

[4] **LMS** (BS2000)
SDF-Format
Benutzerhandbuch

[5] **BS2000/OSD-BC**
DVS Einführung
Benutzerhandbuch

[6] **BS2000/OSD-BC**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

[7] **BS2000**
Einführung in die XS-Programmierung
(für ASSEMBLER-Programmierer)
Benutzerhandbuch

[8] **BS2000/OSD-BC**
Bindelader-Starter
Benutzerhandbuch

[9] **C (BS2000)**
C-Compiler
Benutzerhandbuch

Stichwörter

&SYSDATE 197
&SYSECT 198
&SYSLIST 200
&SYSMOD 202
&SYSNDX 203
&SYSPARM 206
&SYSTEM 206
&SYSTIME 206
&SYSTSEC 207
&SYSVERM 208
&SYSVERS 208
@-Makros 325
@AND 280, **327**
@BEGI 257, 328
@BEND 257, 328
@BREA 268, 272, **329**
@CAS2 263, **332**
@CASE 260, **330**
@CONDI **334**
@CONEN **335**
@CYCL 268, 270, 272, **338**
@DATA 289, **340**
@DO 266, 274, **347**
@ELSE 258, **348**
@END 283, 284, **349**
@ENTR 283, 284, **350**
@EVTLC **368**
@EVTOE **370**
@EXIT 283, 284, **372**
@FREE 297, **376**
@IF 256, 258, **377**
@ININ **378**
@OF 263, **379**
@OFRE 263, **381**
@OR 280, **382**

@PAR 293, 303, **383**
@PASS 283, 301, 305, **389**
@SETJV **395**
@SETPM **396**
@STXDI **397**
@STXEN **398**
@STXIM **401**
@THEN 258, **402**
@THRU 274, **403**
@TOR 280, **405**
@WHEN 256, 268, 272, **406**
@WHIL 256, 266, **407**

A
A-Konstante 80
Abbrechen einer Schleife 329
abschneiden von Konstanten 60
absoluter Ausdruck **17**
absolutes Element 20
ACTR-Anweisung 222
ACTR-Operand 222
ACTR-Zähler 222
Adreßkonstante 80
Adreßpegel
  Ausrichtung mit der CNOP-Anweisung 49
  Bezugnahme 25
  Maximalwert 25
Adreßpegel im Programmabschnitt 36
Adreßpegel setzen 115
Adreßpegel von Namen 10
Adresse 33
  externe 42
  nicht-symbolische 33
  symbolische 33
Adressierungsmodus 41
  Voreinstellung 47
Adressierungsmodus zuordnen 46
ändern des mnemotechnischen Operationscode 112
äußerer Makroaufruf 218
AGO-Anweisung 226
AIF-Anweisung 223
Aktualparameter 303, 305, 307
Alternative (syn) → Entscheidung 258
alternative Formatdarstellung 220

alternatives Anweisungsformat 220
AMODE-Anweisung 46
Anfangsanweisung 144, **234**
Anfangsspalte 7
  Voreinstellung ändern 109
ANOP-Anweisung 229
Anzahlmerkmal, Bezugnahme 183
Anzeige-setzender Assemblerbefehl 276, 280
arithmetische Operatoren 14
arithmetischer Ausdruck 14
arithmetischer Makroausdruck 165
  Berechnung 166
arithmetischer Vergleich 167
Assembler-Quellprogramm 5, 35
Assemblerbefehl, Anzeige-setzender 276
Assemblerbefehle 1
  Format 411
auffüllen von Konstanten 60
Ausdruck 14
  absoluter **17**
  arithmetischer 14
  Berechnung 16
  einfacher 14
  Elemente 14, 20
  Makrosprache 159
  relativer 17, **18**
  Werte 15
ausführbarer Programmabschnitt 36
Ausrichtung
  CNOP-Anweisung 49
  erzwungene 91
Ausrichtung von Konstanten 59
Ausrichtung von Speicherbereichen 90
Auswahlstrukturblock 258
Automatic-Bereich 289, **293**, 341
  Initialisierung 295

B

B-Konstante 70
B-Prozedur 287, 365
BASE-Operand
  @ENTR 365
  CLASS=A 295, 341
  CLASS=B 299, 345

CLASS=C 297, 341
CLASS=S 291, 343
Based-Bereich 289, **299**, 345
Basisadreßregister
Automatic-Bereich 295
Based-Bereich 299
Controlled-Bereich 297
LOCAL-Bereich 293
Prozedur- 285, 287
SAVAREA 293
STACK 293, 301
Static-Bereich 291
Basisadreßregister freigeben 95
Basisadreßregister zuweisen 131
Basisadresse angeben 131
Basisprozedur 287
bedingte EXTRN-Adresse 136
bedingte Verzweigung 223
Bedingung 256
einfache 276
zusammengesetzte 280
Bedingungsmaske 279
Bedingungssymbol 277
benutzereigenes 279
vordefiniertes 277
Befehlssatz
BS2000-ESA 411
BS2000-NXS 411
BS2000-XS 411
Bemerkung 30
Bemerkungseintrag 30
Bemerkungszeile 9
benutzereigenes Bedingungssymbol 279
Bezugnahme auf das Anzahlmerkmal 183
Bezugnahme auf das Definitionsmerkmal 185
Bezugnahme auf das Ganzzahligkeitsmerkmal 181
Bezugnahme auf das Längenmerkmal 26
Makrosprache 179
Bezugnahme auf das Skalenfaktormerkmal 180
Bezugnahme auf das Zählermerkmal 182
Bezugnahme auf den Adreßpegel 25
Bezugnahme auf ein Merkmal, Makrosprache 171
Bibliothekselement kopieren 52
binärer selbstdefinierender Wert 23

Binärkonstante 70
Blockkonzept 253, **256**

C
C-Konstante 68
CASE-Register → Fallunterscheidung durch Nummer 260
CLASS=A 295, 341
CLASS=B 299, 345
CLASS=C 297, 341
CLASS=S 291, 343
CNOP-Anweisung 48
Code-Section (syn) → ausführbarerProgrammabschnitt 36
COM-Anweisung 38, **50**
Contingency-Routine 322
Contingency-Routine abmelden 334
Contingency-Routine anmelden 335
Controlled-Bereich 289, **297**, 341
  Initialisierung 297
COPY-Anweisung 52
COPY-Element, Schachtelungstiefe 418
COPY-Elemente (syn) → Bibliothekselement kopieren 52
CSECT (syn) → ausführbarer Programmabschnitt 37
CSECT-Anweisung 37, **54**
CXD-Anweisung 39, **56**
CYCLE-Schleife (syn) → Schleife mit freier Endebedingung 268

D
D' → Bezugnahme auf das Definitionsmerkmal 185
D-Konstante 74
D-Prozedur 288, 367
Datenbereich 289
  Klasse Automatic 289, **293**
  Klasse Based 290, **299**
  Klasse Controlled 290, **297**
  Klasse Static 289, **291**
Datenklasse (syn) → Datenbereich 289
Datenkonzept 254, **289**
Datenmodul 291
  externer 291
DC-Anweisung 57
definieren von Konstanten 57
definieren von Makronamen → Musteranweisung 144, **209**
definieren von Namen 11
definieren von SET-Parametern 230, 232
Definitionsmerkmal, Bezugnahme 185

dezimaler selbstdefinierender Wert 22
Dezimalkonstante 78
DROP-Anweisung 34, **95**
DS-Anweisung 87
  Längenberechnung 91
DSECT (syn) → Pseudoabschnitt 37
DSECT-Anweisung 37, **96**
Dummy-Section (syn) → Pseudoabschnitt 37
DXD-Anweisung 40, **93**
dynamische Parameterübergabe 301, **305**
dynamisches Prozedurende 283, 284, 372

E
E-Konstante 74
E-Prozedur 285, 358
einfache Bedingung 276
einfacher Ausdruck 14
Eingabeformat 109
Eingabesatz 109, **418**
  Länge 418
  Voreinstellung 418
Einsprungstelle 42
Einsprungstelle kennzeichnen 103
Einträge
  Instruktion 7
  Instruktion der Makrosprache 147
  XREF-Listing 418
EJECT-Anweisung 100
Element
  absolutes 20
  relatives 20
Elemente von Ausdrücken 14, **20**
END-Anweisung 101
Endanweisung, Makrosprache 144, **234**
Ende der Übersetzung 101
Endebedingung 268, 272
Endspalte 7
  Voreinstellung ändern 109
ENTRY-Adresse 42
ENTRY-Adresse kennzeichnen 103
ENTRY-Anweisung 103
Entscheidung **258**, 348, 377, 402
EQU-Anweisung 106
Ereignisbehandlung, ILCS 321, 322

erster Programmabschnitt 36
  kennzeichnen 126
ESID-Nummern 419
explizite Längenangabe, EQU-Anweisung 106
Exponentenfaktor 66
externe Adresse 42
externe Adresse kennzeichnen 104
externe Prozedur 284, 285, 287
externer Datenmodul 291
externer Pseudoabschnitt 38
  Definition 137
  Referenz 137
externer Static-Bereich 291
EXTRN-Adresse 42
  bedingte 136
EXTRN-Adresse kennzeichnen 104
EXTRN-Anweisung 104

F

F-Konstante 71
Fallunterscheidung durch Nummer 260, 330
Fallunterscheidung durch Vergleich 262, 332, 379, 381
Festpunktkonstante 71
Folgesymbol 149
  generiertes Format 158
  Namenseintrag 149
  Operandeneintrag 158
  Standardformat 149
  vordefinierte Makros 325
Folgezeile kopieren 120
Formalparameter 303, 305, 307, 313, 315
Fortsetzungsspalte 7, 31
  Voreinstellung ändern 109
Fortsetzungszeichen 31
Fortsetzungszeichenspalte 31
Fortsetzungszeile 31
  Bemerkungseintrag 30
  Kommentar 9
Freigabe von Speicherbereich 297, 376
freigeben von Basisadreßregistern 95

G
Ganzzahligkeitsmerkmal, Bezugnahme 181
GBLx-Anweisung 230
gemeinsamer Hilfsabschnitt 38
  definieren 50
generierter Kennwortoperand 213
generierter Makroname 210, 211
generierter Stellungsoperand 213
generierter variabler Parameter 153
Gleichsetzen 106
Gleitpunktkonstante 74
  Charakteristik 74
  Mantisse 74
  Maschinenformat 74
globalen SET-Parameter definieren 230
globaler SET-Parameter 191
globaler variabler Systemparameter 196
Größe eines Objektmoduls 420
Groß- und Kleinschreibung 7
Großbuchstaben 7
GSEQ-Anweisung 450

H
H-Konstante 71
Hauptprozedur 284, 285, 352
Hilfsabschnitt
  gemeinsamer 38
  gemeinsamer, definieren 50
Hochkomma 5
Hochkomma in C-Konstanten 68
Hochkomma in Makroaufrufen 212
Hochkomma in selbstdefinierenden Zeichenwerten 24
Hochkomma in Zeichenwerten 162

I
I' → Bezugnahme auf das Ganzzahligkeitsmerkmal 181
I-Prozedur 285, 358
ICTL-Anweisung 109
IDA 440
ILCS bei nachgeladenen Moduln aufrufen 378
ILCS-Schnittstelle **316**
implizite Länge
  DC-Konstanten 63
  DS-Anweisung 89
  EQU-Anweisung 106

indizierter SET-Parameter 194
Initialisierung von Automatic Bereichen 295
Initialisierung von Controlled-Bereichen 297
innere Makrodefinition 143, 146
innerer Makroaufruf 218
Instruktion 7
  Einträge 7
  Makrosprache 147
interne Prozedur 284, 285, 287
interner Static-Bereich 291
ISEQ-Anweisung 442
ISLU 440
iterative Schleife 274, 347, 403

J
Ja-Unterblock 259, 402

K
K' → Bezugnahme auf das Zählermerkmal 182
Kennwortoperand 187, **213**
Klammern in arithmetischen Ausdrücken 16
Klammern in Makroaufrufen 212
Kleinbuchstaben 7
Kommas in Makroaufrufen 212
Kommentar 9
  Makrosprache 147
Kommentarzeile 9
Komparand 262, 264, 332, 379
Konstante
  Ausrichtung 59
  Bezugnahme auf den Adreßpegel 61
  Exponentenfaktor 66
  implizite Länge 63
  Längenfaktor 63
  Skalenfaktor 64
  Speicherplatz 60
Konstante abschneiden 60
Konstante auffüllen 60
Konstante definieren 57
Konstantentyp 58, **67**
kopieren Bibliothekselement 52
kopieren Folgezeile 120
kopieren Text 119

L
L' → Bezugnahme auf das Längenmerkmal 26, 179
L-Konstante 74
L-Prozedur 287, 365
Ladeattribut 41
  Voreinstellung 122
Ladeattribut zuordnen 121
Länge von Namen 10, 417
Längenfaktor
  DC-Anweisung 63
  DS-Anweisung 89
  DXD-Anweisung 94
Längenmerkmal
  Bezugnahme 26
  Bezugnahme in der Makrosprache 179
Längenmerkmal von Namen 11
Laufanweisung (syn) → iterative Schleife 274
Laufvariable 274, 403
Laufzeitsystem 254, 285
Layout der Kontext-Beschreibung definieren 368
Layout der Unterbrechungsmeldung definieren 401
LCLx-Anweisung 232
Literal 27
  Format 27
  maximale Anzahl 417
  Regeln 28
Literalbereich 29
  Lage festlegen 110
Literalbereich definieren 110
LLM-Format
  PUNCH-Anweisung 119
  REPRO-Anweisung 120
LOCAL-Bereich 293, 315
LOCAL-Operand 293, 315, 353, 360
logische Operatoren 169
logischer Ausdruck 169
logisches Oder 280, 382, 405
logisches Oder mit Prorität 280
logisches Und 280, 327
lokalen SET-Parameter definieren 232
lokaler Pseudoabschnitt 293, 315, **388**
lokaler SET-Parameter 191
  implizit vereinbarter 194
lokaler variabler Systemparameter 196

Longjump, ILCS-Schnittstelle 318
Low-level-Prozedur 287
LTORG-Anweisung 110, 349

M
M-Prozedur 285, 352
MACRO-Anweisung 234
Macro-trace 239
Macro-trace beenden 241
Makroauflösung → Makrogenerierung 142
Makroaufruf 142, **209**
  äußerer 218
  innerer 218
  Länge der Operanden 418
  Namenseintrag 211
  Operandeneintrag 212
  Operandenunterliste 215
  Operationseintrag 211
  Regeln für Operanden 212
Makroausdruck 159
  arithmetischer 165
  logischer 169
  Vergleichsausdruck 167
Makrobibliothek 144
Makrodefinition 142
  Aufbau 144
  innere 143, 146
  maximale Anzahl 418
  Schachtelungstiefe 418
Makrodefinition im Quellprogramm 143
Makrogenerierung 142
Makrogenerierung beenden → MEXIT-Anweisung 235
Makros, vordefinierte 254, **325**
Maschineninstruktion (syn) → Assemblerbefehle 1
MAXPRM-Operand 305, 353, 360
MCALL-Anweisung 450
MCALL-Option 440
Mehrfachverzweigung 330, 332
MEND-Anweisung 234
Merkmal, Bezugnahme 171
Merkmale von Programmabschnitten 41
MEXIT-Anweisung 235
mnemotechnischen Operationscode ändern 112
mnemotechnischen Operationscode zuweisen 112

mnemotechnischer Operationscode, Assemblerbefehle 411
MNOTE-Anweisung 237
Modellanweisung 144
Modifizierfaktor
  DC-Anweisung 58
  DS-Anweisung 88
Modulgröße 420
Monitorjobvariable 324
Monitorjobvariable setzen 395
MTRAC-Anweisung 239
Musteranweisung 144, **209**
  Namenseintrag 211
  Operandeneintrag 212
  Operationseintrag 211

N

N' → Bezugnahme auf das Anzahlmerkmal 183
Name **10**, 20
  absoluter Wert 10
  Adreßpegel 10
  Länge 10, 417
  Längenmerkmal 11
  maximale Anzahl 417
  Merkmale in der Makrosprache 172
  Regeln 10
  reservierter 326
  Unterstrich 10
  unzulässige Angaben 11
  vordefinierter 326, 420
  Wert 10
  zulässige Angaben 11
Name definieren 11
Namenseintrag 10
  Makrosprache 149
  unzulässige Angaben 11
  vordefinierte Makros 325
  zulässige Angaben 11
Nassi-Shneiderman-Diagramm 256
Nein-Unterblock 259, 348
Nicht-STXIT-Ereignis signalisieren 370
nicht-symbolische Adresse 33
NTRAC-Anweisung 241
Nulloperation (ANOP) 229
Nulloperation setzen (CNOP) 48

O
Objektmodul, Größe 420
Oder, logisches 280, 382, 405
Operandeneintrag 13
  Literal 27
  Makrosprache 157
  vorfdefinierte Makros 325
Operandenunterliste 215
Operationscode, OPSYN-Anweisung 112
Operationseintrag 12
  Makrosprache 155
  vordefinierte Makros 325
  zulässiger 12
Operatoren
  arithmetische 14
  logische 169
  Vergleichs- 167
OPSYN-Anweisung 112
OPTIMAL-Schnittstelle, Parameterübergabe 307
ORG-Anweisung 115

P
P-Konstante 78
paarweises Auftreten von relativen Elementen 18
PAGE → Merkmale von Programmabschnitten 41
PAR-Operand, @PASS 303, 390
Parameter 301
Parameterliste 301, 302, 303, 305, 383
  Adresse 309
Parameterübergabe 284, **301**
  dynamisch 301, **305**
  ILCS-Schnittstelle 319
  OPTIMAL 301, **307**
  STANDARD 301
  STANDARD-Schnittstelle 303
  statisch 301, **303**
Parameterübernahme 309
  Formalparameter 313, 315
  STANDARD-Schnittstelle 309
PASS-Operand
  @ENTR 313, 315, 359
  @PASS 305, 307, 393
PASS-operand, @ENTR 311
PLIST-Operand

@ENTR 313, 315, 359
@PAR 303, 315, 383, 386
@PASS 305, 307, 393
PRINT-Anweisung 117
PRINT-Parameter 117
PRINT-Parameter sichern 124
PRINT-Parameter zurücksetzen 129
Programm → Übersetzungseinheit 35
Programmabschnitt 35, **36**
Adreßpegel 36
Adressierungsmodus 41
ausführbarer 36
erster 36
Ladeattribut 41
Literalbereich 29, 36
Merkmale 41
Referenz- 37
Programmabschnitt definieren 54
Programmabschnitt fortsetzen 54
Programmanfang definieren 126
Programmaske 324
Programmaske setzen oder rücksetzen 396
Programmentwurf 254
Programmverknüpfung, symbolische 42
Prozedur 282
Aufruf 282, **389**
Basisadreßregister 285, 287
externe 284, 285, 287
Hauptprozedur 285
interne 284, 285, 287
reentrant-fähige 282, 289
Typ B 287, 365
Typ D 288, 367
Typ E 285, 358
Typ I 285, 358
Typ L 287, 365
Typ M 285, 352
Verkettungsinformation 301
Prozedur-STACK 293, 301
Prozeduranfang 283, 350
Prozedurende 284
dynamisches 283, 284, 372
statisches 283, 284, 349
Prozedurerklärung 284, 350

Prozedurkonzept 253, **282**
Prozedurkopf 284, 350
Prozedurtyp 285
Prozedurverknüpfung 301
  ILCS-Schnittstelle 317
PRVLGD → Merkmale von Programmabschnitten 41
Pseudoabschnitt 37
  Definition für Parameterübernahme 386
  externer 38
  externer, Definition 137
  externer, Referenz 137
  lokaler 293, 315, **388**
Pseudoabschnitt definieren 96
Pseudoabschnitt fortsetzen 96
Pseudocode 254
Pseudoprozedur 288
Pseudoregister 39
  Adressierung 40
  DSECT-Anweisung 97
  USING-Anweisung 135
  Verwendung 39
Pseudoregister definieren 40, 93
Pseudoregistervektor 39
  Länge 56
  Speicherplatz reservieren 56
PUBLIC → Merkmale von Programmabschnitten 41
PUNCH-Anweisung 119
  LLM-Format 119

Q

Q-Konstante 40, **86**
Quellprogramm 1, 5, 35
Quellprogrammtext 1, 5, 35

R

READ → Merkmale von Programmabschnitten 41
redefinieren von Speicherbereichen 91
reentrant-fähige Prozedur 282, 289
Referenz-Programmabschnitt 37
Registerkonventionen 420
  ILCS-Schnittstelle 317
Registersicherung 282, 285, 301
relative Elemente, paarweises Auftreten 18
relativer Ausdruck 17, **18**
relatives Element 20

REPRO-Anweisung 120
  LLM-Format 120
reservieren von Speicherplatz 87
reservierte Namen 326
RESIDENT → Merkmale von Programmabschnitten 41
Residenzmodus (syn) → Ladeattribut 41
Rest-Unterblock 262, 381
restaurieren PRINT-Parameter 129
restaurieren USING-Status 129
RMODE-Anweisung 121
Rückkehrwert 284, 373

S

S' → Bezugnahme auf das Skalenfaktormerkmal 180
S-Konstante 82
SAVAREA 301
Schachtelung von Strukturblöcken 256
Schachtelungstiefe
  COPY-Anweisungen 52, 418
  Makroaufrufe **218**, 418
  Makrodefinitionen 143
Schleife 266
  abbrechen 329
  Ausgang 329
  iterative 274, 347, 403
  Kopf 338
  Unterblock 347
  Zählschleife 270
  Zählschleife mit freier Endebedingung 272
Schleife mit freier Endebedingung 268, 338, 406
Schleife mit Vorabprüfung 266, 347, 407
Schleifen-Unterblock 266
Schrittweite 274, 403
sedezimaler selbstdefinierener Wert 23
Sedezimalkonstante 69
Seitenüberschrift 128
Seitenvorschub 100
selbstdefinierender Wert 21
  binärer 23
  dezimaler 22
  sedezimaler 23
selbstdefinierender Zeichenwert 24
Selektor 262, 264, 332, 379
Sequenz 257

SET-Parameter 152, **189**
  globaler 191
  indizierter 194
  lokaler 191
  Standardwerte → vordefinierte SET-Parameter 190
SETA-Anweisung 243
SETA-Parameter 189
SETA-Parameter setzen 243
SETB-Anweisung 245
SETB-Parameter 189
SETB-Parameter setzen 245
SETC-Anweisung 247
SETC-Parameter 189
SETC-Parameter setzen 247
setzen Adreßpegel 115
setzen arithmetischen Wert 243
setzen Binärwert 245
setzen Nulloperation 48
setzen Zeichenwert 247
sichern PRINT-Parameter 124
sichern USING-Status 124
Skalenfaktor 64
  Festpunktkonstanten 65
  Gleitpunktkonstanten 65
Skalenfaktormerkmal, Bezugnahme 180
Source (syn) → Quellprogramm 1, 5, 35
Source-Deck-Makro (syn) → Makrodefinition im Quellprogramm 143
SPACE-Anweisung 123
Speicheranforderung 340
Speicherbereich, explizite Freigabe 297, 376
Speicherbereich redefinieren 91
Speicherklasse 289, 340
Speicherplatz reservieren 87
Speicherverwaltung 254, 289
Sprachinitialisierung, ILCS 324
Sprungziel definieren 229
STACK, Prozedur- 293, 301
STACK-Anweisung 124
Standard-Contingency-Handler (SCH), ILCS 322
Standard-Event-Handler (SEH), ILCS 321
STANDARD-Schnittstelle
  Parameterübergabe 303
  Parameterübernahme 309
Standard-STXIT-Handler (SSH), ILCS 323

START-Anweisung 36, **126**
Statementnummer 417
Static-Bereich 289, **291**, 343
  externer 291
  interner 291
statische Parameterübergabe 301, **303**
statisches Prozedurende 283, 284, 349
Stellungsoperand 187, **213**
Stern-Adresse (syn) → Bezugnahme auf den Adreßpegel 25
Steuerfluß 254, 256
Struktogramm 256
Strukturblock 256, 282
  Entscheidung 258
  Fallunterscheidung durch Nummer 260
  Fallunterscheidung durch Vergleich 262
  iterative Schleife 274
  Schachtelung 256
  Schleife mit freier Endebedingung 268
  Schleife mit Vorabprüfung 266
  Sequenz 257
  Zählschleife 270
  Zählschleife mit freier Endebedingung 272
Strukturblock-Ende 328
Strukturierte Programmierung **253**
  Blockkonzept 256
  Datenkonzept 289
  Einführung 253
  ILCS-Schnittstelle 316
  Prozedurkonzept 282
  Prozedurverknüpfung und Parameterübergabe 301
  vordefinierte Makros 325
STXIT-Behandlungsroutine abmelden 397
STXIT-Behandlungsroutine anmelden 398
STXIT-Ereignisse 321, 323
STXIT-Routine 323
Sublist (syn) → Operandenunterliste 215
Suboperand → Operandenunterliste 215
symbolische Adresse 33
symbolische Programmverknüpfung 42
symbolischer Parameter 152, **187**

T
T' → Bezugnahme auf das Typenmerkmal 174
Teilzeichenfolge 163
  Verkettung 164
Text kopieren 119
textuelle Ersetzung 151
TITLE-Anweisung 128
Typenmerkmal, Bezugnahme 174

U
Übergabe von Parametern 284, **301**
Übernahme von Parametern 309
Übersetzungseinheit 1, 35
  Programmabschnitt 35
Übersetzungsende 101
Übersetzungsprotokoll
  Inhalt 117
  Seitenüberschrift 128
  Seitenvorschub 100
  Zeilenvorschub 123
Übersetzungsprotokoll ausdrucken 117
überwachen Verzweigungen 239
Überwachung beenden 241
unbedingte Verzweigung 226
Und, logisches 280, 327
Und-Zeichen 5
Und-Zeichen in C-Konstanten 68
Und-Zeichen in Makroaufrufen 212
Und-Zeichen in selbstdefinierenden Zeichenwerten 24
Und-Zeichen in Zeichenwerten 162
UNSTK-Anweisung 129
Unterblock 258
  Rest- 262
  Rest-Unterblock 381
  Schleifen- 266
Unterliste → Operandenunterliste 215
Unterstrich 5
Unterstrich in Namen 10
USING-Anweisung 34, **131**
USING-Status sichern 124
USING-Status zurücksetzen 129

V
V-Konstante 42, **84**
variabler Parameter 151
  Bemerkungseintrag 151
  generierter 153
  Kommentarzeile 148
  Merkmale 173
  Namenseintrag 151
  Operandeneintrag 157
  Operationseintrag 155
  Verkettung 154
  vordefinierte Makros 325
variabler Systemparameter 152, **196**
  globaler 196
  lokaler 196
Vergleich
  arithmetischer 167
  Zeichen- 167
Vergleichsausdruck 167
Vergleichsbefehl 264
Vergleichsoperatoren 167
Verkettung von variablen Parametern und alphanumerischen Zeichen 154
Verkettung von Zeichenwerten und Teilzeichenfolgen 164
Verkettungsinformation 301
Verknüpfung von Programmen 42
Verknüpfung von Prozeduren 301
Verzweigung 377
  bedingte 223
  unbedingte 226
Verzweigungen überwachen 239
VLIST-Operand, @PAR 303, 383
vordefinierte Makros 254, **325**
vordefinierte Namen 326, 420
vordefinierte SET-Parameter 190
vordefiniertes Bedingungssymbol 277

W
Wert, selbstdefinierender 21
Wert von Ausdrücken 15
Wert von Namen 10
WHILE-Schleifen (syn) → Schleife mit Vorabprüfung 266
Wiederholungsfaktor
  DC-Anweisung 58
  DS-Anweisung 88

DXD-Anweisung 93
Zählschleife 270, 272
WXTRN-Anweisung 136

X
X-Konstante 69
XDSEC (syn) → externer Pseudoabschnitt 38
XDSEC-Anweisung 38, **137**
XREF-Listing 418

Y
Y-Konstante 80

Z
Z-Konstante 78
zählen Verzweigungen 222
Zählermerkmal, Bezugnahme 182
Zählschleife 270, 338
Zählschleife mit freier Endebedingung 272, 338, 406
Zeichenausdruck 161
Zeichenkonstante 68
Zeichenvergleich 167
Zeichenvorrat 5
Zeichenwert 161
selbstdefinierender 24
Verkettung 164
Zeilennummer → Statementnummer 417
Zeilenvorschub 123
zurücksetzen PRINT-Parameter 129
zurücksetzen USING-Status 129
zusammengesetzte Bedingung 280
zuweisen Basisadreßregister 131
zuweisen des mnemotechnischen Operationscode 112
zuweisen von Werten 106

Inhalt

1	**Einleitung**	**1**
1.1	Kurzbeschreibung des Produkts	1
1.2	Zielgruppe	2
1.3	Konzept des Handbuchs	2
1.4	Änderungen gegenüber der vorigen Ausgabe	3
1.5	Metasprache	4
2	**Struktur der Assemblersprache**	**5**
2.1	Zeichenvorrat	5
2.2	Instruktionen und Kommentare	7
2.3	Namenseintrag	10
	Definieren von Namen	11
2.4	Operationseintrag	12
2.5	Operandeneintrag	13
2.5.1	Ausdrücke	14
2.5.1.1	Einfache Ausdrücke	14
2.5.1.2	Arithmetische Ausdrücke	14
2.5.1.3	Absolute und relative Ausdrücke	17
2.5.2	Elemente von Ausdrücken	20
2.5.2.1	Namen	20
2.5.2.2	Selbstdefinierende Werte	21
2.5.2.3	Bezugnahme auf den Adreßpegel	25
2.5.2.4	Bezugnahme auf das Längenmerkmal	26
2.5.3	Literale	27
2.6	Bemerkungseintrag	30
2.7	Fortsetzungszeichen	31
3	**Adressierung, Programmunterteilung und Programmverknüpfung**	**33**
3.1	Adressierung	33
3.2	Programmunterteilung	35
3.3	Programmabschnitte	36
3.3.1	Ausführbare Programmabschnitte	36
3.3.2	Referenz-Programmabschnitte	37
	Pseudoabschnitt	37
	Externer Pseudoabschnitt	38

	Gemeinsamer Hilfsabschnitt	38
	Pseudoregister	39
3.3.3	Merkmale von Programmabschnitten	41
3.4	Symbolische Programmverknüpfung	42
4	**Assembleranweisungen**	**43**
4.1	Allgemeines	43
4.2	Beschreibung der Anweisungen	46
	AMODE Adressierungsmodus zuordnen	46
	CNOP Nulloperation setzen	48
	COM Gemeinsamen Hilfsabschnitt definieren	50
	COPY Quellprogrammtext aus Bibliothekselement kopieren	52
	CSECT Programmabschnitt definieren	54
	CXD Speicherplatz für die Länge des Pseudoregistervektors reservieren	56
	DC Konstante definieren	57
	Modifizierfaktoren	63
	Längenfaktor	63
	Skalenfaktor	64
	Exponentenfaktor	66
	Konstantentypen	67
	Zeichenkonstanten C	68
	Sedezimalkonstanten X	69
	Binärkonstanten B	70
	Festpunktkonstanten F und H	71
	Gleitpunktkonstanten E, D und L	74
	Dezimalkonstanten P und Z	78
	Adreßkonstanten	80
	Adreßkonstanten vom Typ A und Y	80
	Adreßkonstanten vom Typ S	82
	Adreßkonstanten vom Typ V	84
	Adreßkonstanten vom Typ Q	86
	DS Speicherplatz reservieren	87
	DXD Pseudoregister definieren	93
	DROP Basisadreßregister freigeben	95
	DSECT Pseudoabschnitt definieren	96
	EJECT Seitenvorschub	100
	END Übersetzungsende	101
	ENTRY ENTRY-Adresse kennzeichnen	103
	EXTRN EXTRN-Adresse kennzeichnen	104
	EQU Gleichsetzen	106
	ICTL Eingabeformat steuern	109
	LTORG Literalbereich definieren	110

	OPSYN Mnemotechnischen Operationscode zuweisen	112
	ORG Adreßpegel setzen	115
	PRINT Protokollinhalt steuern	117
	PUNCH Text in Objektmodul kopieren	119
	REPRO Folgezeile in Objektmodul kopieren	120
	RMODE Ladeattribut zuordnen	121
	SPACE Zeilenvorschub	123
	STACK USING- oder PRINT-Status sichern	124
	START Programmanfang definieren	126
	TITLE Protokollüberschrift	128
	UNSTK USING- oder PRINT-Status restaurieren	129
	USING Basisadreßregister zuweisen	131
	WXTRN Bedingte EXTRN-Adresse kennzeichnen	136
	XDSEC Externen Pseudoabschnitt definieren	137
5	**Struktur der Makrosprache**	**141**
5.1	Aufruf und Definition von Makros	142
5.1.1	Ablage der Makrodefinition	143
5.1.2	Aufbau der Makrodefinition	144
5.1.3	Aufbau der inneren Makrodefinition	146
5.2	Instruktionen und Kommentare	147
5.3	Namenseintrag	149
5.3.1	Folgesymbole	149
5.3.2	Variable Parameter	151
5.3.3	Generierte variable Parameter	153
5.3.4	Verkettung von variablen Parametern und alphanumerischen Zeichen	154
5.4	Operationseintrag	155
	Variable Parameter im Operationseintrag	155
5.5	Operandeneintrag	157
5.5.1	Variable Parameter im Operandeneintrag	157
5.5.2	Folgesymbole im Operandeneintrag	158
5.5.3	Makroausdrücke	159
5.5.4	Zeichenausdrücke	161
5.5.4.1	Zeichenwert	161
5.5.4.2	Teilzeichenfolge	163
5.5.4.3	Verkettung von Zeichenwerten und Teilzeichenfolgen	164
5.5.5	Arithmetische Makroausdrücke	165
5.5.6	Vergleichsausdrücke	167
5.5.7	Logische Ausdrücke	169
5.5.8	Bezugnahme auf Merkmale	171
5.5.8.1	T' Bezugnahme auf das Typenmerkmal	174
5.5.8.2	L' Bezugnahme auf das Längenmerkmal	179
5.5.8.3	S' Bezugnahme auf das Skalenfaktormerkmal	180
5.5.8.4	I' Bezugnahme auf das Ganzzahligkeitsmerkmal	181

5.5.8.5 K' Bezugnahme auf das Zählermerkmal . . . 182
5.5.8.6 N' Bezugnahme auf das Anzahlmerkmal . . . 183
5.5.8.7 D' Bezugnahme auf das Definitionsmerkmal . . . 185

6 Variable Parameter . . . 187
6.1 Symbolische Parameter . . . 187
6.2 SET-Parameter . . . 189
Globale und lokale SET-Parameter . . . 191
Implizit vereinbarte lokale SET-Parameter . . . 194
Indizierte SET-Parameter . . . 194
6.3 Variable Systemparameter . . . 196
&SYSDATE . . . 197
&SYSECT . . . 198
&SYSLIST . . . 200
&SYSMOD . . . 202
&SYSNDX . . . 203
&SYSPARM . . . 206
&SYSTEM . . . 206
&SYSTIME . . . 206
&SYSTSEC . . . 207
&SYSVERM . . . 208
&SYSVERS . . . 208

7 Instruktionen der Makrosprache . . . 209
7.1 Musteranweisung und Makroaufruf . . . 209
7.1.1 Kennwort- und Stellungsoperanden . . . 213
7.1.2 Operandenunterlisten . . . 215
7.1.3 Äußere und innere Makroaufrufe . . . 218
7.1.4 Alternatives Anweisungsformat . . . 220
7.2 Beschreibung der Makroanweisungen . . . 222
ACTR Verzweigungen zählen . . . 222
AIF Bedingt verzweigen . . . 223
AGO Unbedingt verzweigen . . . 226
ANOP Nulloperation . . . 229
GBLx Globalen SET-Parameter definieren . . . 230
LCLx Lokalen SET-Parameter definieren . . . 232
MACRO Anfangsanweisung . . . 234
MEND Endanweisung . . . 234
MEXIT Ausgang aus einer Makrodefinition definieren . . . 235
MNOTE Meldungen absetzen . . . 237
MTRAC Macro trace . . . 239
NTRAC Macro trace beenden . . . 241
SETA SETA-Parameter setzen . . . 243
SETB SETB-Parameter setzen . . . 245
SETC SETC-Parameter setzen . . . 247

8	**Makrosprachelemente im Assembler-Quellprogrammtext**	**249**
9	**Strukturierte Programmierung mit ASSEMBH**	**253**
9.1	Einführung	253
9.2	Blockkonzept	256
9.2.1	Sequenz	257
9.2.2	Auswahlstrukturblöcke	258
9.2.2.1	Entscheidung	258
9.2.2.2	Fallunterscheidung durch Nummer	260
9.2.2.3	Fallunterscheidung durch Vergleich	262
9.2.3	Schleifen	266
9.2.3.1	Schleife mit Vorabprüfung	266
9.2.3.2	Schleife mit freier Endebedingung	268
9.2.3.3	Zählschleife	270
9.2.3.4	Zählschleife mit freier Endebedingung	272
9.2.3.5	Iterative Schleife	274
9.2.4	Einfache Bedingungen	276
9.2.4.1	Vordefinierte Bedingungssymbole	277
9.2.4.2	Benutzereigene Bedingungssymbole	279
9.2.5	Zusammengesetzte Bedingungen	280
9.3	Prozedurkonzept	282
9.3.1	Prozedurerklärung und Prozedurende	283
9.3.2	Prozedurtypen	285
9.3.2.1	Prozeduren der Typen M, E und I	285
9.3.2.2	Prozeduren der Typen B, L und D	287
9.4	Datenkonzept	289
9.4.1	Datenbereiche der Klasse Static	291
9.4.2	Datenbereiche der Klasse Automatic	293
9.4.3	Datenbereiche der Klasse Controlled	297
9.4.4	Datenbereiche der Klasse Based	299
9.5	Prozedurverknüpfung und Parameterübergabe	301
9.5.1	Parameterübergabe über die STANDARD-Schnittstelle	303
9.5.1.1	Statische Parameterübergabe	303
9.5.1.2	Dynamische Parameterübergabe	305
9.5.2	Parameterübergabe über die OPTIMAL-Schnittstelle	307
9.5.3	Parameterübernahme	309
9.5.3.1	Parameterübernahme über die STANDARD-Schnittstelle	309
9.5.3.2	Parameterübernahme über die OPTIMAL-Schnittstelle	311
9.5.3.3	Parameterübernahme in Formalparameter	313
9.5.3.4	Parameterübernahme in Formalparameter im LOCAL-Bereich	315
9.6	ILCS-Anschluß für die Strukturierte Programmierung	316

9.6.1 Prozedurverknüpfung . . . 317
Registerkonventionen (ILCS- und Nicht-ILCS-Schnittstelle) . . . 317
Parameterübergabe . . . 319
9.6.2 Benutzereigene Routinen anmelden . . . 321
9.6.3 Ereignisse behandeln (Event Handling) . . . 321
9.6.4 Folgeprozesse behandeln (Contingency Handling) . . . 322
9.6.5 Unterbrechungsereignisse behandeln (STXIT Handling) . . . 323
9.6.6 Programmaske setzen . . . 324
9.6.7 MONJV-Wert in der PCD setzen . . . 324
9.6.8 Sprachinitialisierung bei nachgeladenen Moduln . . . 324

10 Vordefinierte Makros für die Strukturierte Programmierung . . . 325
Allgemeine Programmierhinweise . . . 325
@AND Logisches 'Und' . . . 327
@BEGI Sequenz . . . 328
@BEND Strukturblock-Ende . . . 328
@BREA Abbrechen einer Schleife . . . 329
@CASE Fallunterscheidung durch Nummer . . . 330
@CAS2 Fallunterscheidung durch Vergleich . . . 332
@CONDI Contingency-Routine abmelden . . . 334
@CONEN Contingency-Routine anmelden . . . 335
@CYCL Schleifen-Kopf . . . 338
@DATA Datenzugriff und Speicheranforderung . . . 340
@DO Schleifen-Unterblock . . . 347
@ELSE Nein-Unterblock . . . 348
@END Statisches Prozedurende . . . 349
@ENTR Prozeduranfang . . . 350
@EVTLC Layout der Kontext-Beschreibung definieren . . . 368
@EVTOE Nicht-STXIT-Ereignis signalisieren . . . 370
@EXIT Dynamisches Prozedurende . . . 372
@FREE Speicherfreigabe . . . 376
@IF Entscheidung . . . 377
@ININ ILCS bei nachgeladenen Moduln aufrufen . . . 378
@OF Fall-Unterblock . . . 379
@OFRE Rest-Unterblock . . . 381
@OR Logisches 'Oder' . . . 382
@PAR Definition von Bereichen . . . 383
@PASS Prozedur-Aufruf . . . 389
@SETJV Monitorjobvariable setzen . . . 395
@SETPM Programmaske setzen oder rücksetzen . . . 396
@STXDI STXIT-Behandlungsroutine abmelden . . . 397
@STXEN STXIT-Behandlungsroutine anmelden . . . 398
@STXIM Layout der Unterbrechungsmeldung definieren . . . 401

@THEN Ja-Unterblock 402
@THRU Iterative Schleife 403
@TOR Logisches 'Oder mit Priorität' 405
@WHEN Bedingung für Schleifenabbruch 406
@WHIL Schleife mit Vorabprüfung 407

11 Anhang 409
11.1 Zusammenfassung der DC-Konstanten 410
11.2 Format der Assemblerbefehle 411
11.3 Assembler-Restriktionen 417
11.4 Pseudoregister, Beispiele 421
11.5 Parameterübergabe bei der strukturierten Programmierung, Beispiel 425
11.6 Unterschiede von ASSEMBH V1.2A und ASSEMB V30.0A 440

Handbuchergänzungen
Literatur
Stichwörter

FUJITSU

ASSEMBH

Beschreibung

Stand der Beschreibung:
ASSEMBH V1.2
Mit Ergänzungkapitel zu ASSEMBH V1.2D

Ausgabe Juni 2010

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an manuals@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2000

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2000 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.
Das vorliegende Dokument ist eine fachlich ergänzte Neuausgabe eines früheren Handbuchs zu einer bereits vor längerer Zeit freigegebene Produktversion.
Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.
Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form …@ts.fujitsu.com.
Die Internetseiten von Fujitsu Technology Solutions finden Sie unter http://de.ts.fujitsu.com/