

TR 440
TAS – Handbuch
Telefunken-Assemblersprache

Allgemeines		A
Programm	Aufbau	B
	Befehle und Adressierung	C
	Anschluß anderer Programme	D
	Übersetzen, Testen	E
	Konstanten, Pseudobefehle	G
Operationen	Festkommaarithmetik	H
	Gleitkommaarithmetik	K
	Nichtzahlwörter	L
	Adressenrechnung	M
	Sonstige Operationen	N

Änderungsstand

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16		
17	18	19	Änderungsdienst wird nicht mehr durchgeführt.														

TELEFUNKEN
COMPUTER

TAS - HANDBUCH

Allgemeines

STICHWORTVERZEICHNIS

Falls Sie ein Stichwort in dem Verzeichnis vermissen oder Vorschläge in bezug auf Änderungen und Ergänzungen haben, teilen Sie uns dies bitte mit. Wir sind für Ihre Mitarbeit dankbar.

N3/GR/V23
Böckmann
Tel. 474



ABBRUCHKRITERIUM L3-2,M11-1
 ABFRAGEN
 - EINES BITS L6-2
 - DER MARKE H2-4
 - DER TYPENKENNUNG L6-2
 ABGRENZUNGSTEIL B5-1,5-5,C0-1,G2-1
 ABLAGE G2-4
 - DER INFORMATIONSEINHEITEN B3-5
 -BEREICHE G2-4
 -KENNUNG B3-5,G2-5
 -OPTIMIERUNG B3-8
 -SCHLUESSEL B3-5,G2-4
 -SPEZIFIKATION B3-7,5-3
 ABRUFPHASE M1-1
 ABSOLUTE
 - ANGABEN C4-1
 - BEZUEGE B3-12,C2-1,5-3
 - INDEXADRESSEN C6-1
 - INDEXPGEGEL C6-3
 - SPRUENGE N2-2
 - SUMMENANGABEN C4-3
 ABSOLUTES SPRUNGZIEL N2-3
 ACHELSEITE B3-13
 ADDITION H2-5,K3-4,L8-1,10-2
 ADJUNKTION N6-2
 ADRESSEN C2-1,D2-8,3-11,L8-2
 -KONSTANTE B5-4,G1-7
 -ZONE B3-1,3-5,G2-4,2-6
 -, GERADE B5-4,C2-2,L4-1
 -, UNGERADE B5-4,C2-2,L4-1
 ADRESSIERUNGS
 -BEDINGUNGEN B3-10,3-12
 -BEREICHE B3-2
 ALARM H2-2,2-4,11-6
 -MELDUNG N2-7
 - ALS ABBRUCHKRITERIUM M11-6
 ALPHATEXT L0-1,2-1
 AKTUELLER PARAMETER D7-2
 ANTIVALENZ N6-2
 ARBEITSSPEICHER G2-3

 BEDINGTE SPRUENGE N2-2
 BEFEHLE B5-1
 BEFEHLS
 -BEREICH G2-4,B3-3
 -CODE C1-1
 -FOLGE D1-2
 BENENNUNG B5-1
 BENENNUNGSTEIL C0-1
 BENUTZER-BIBLIOTHEK D7-1
 BEREICH B3-2,H2-1,K2-1
 BEREICHSUEBERSCHREITUNG K2-2,K2-4,M1-2
 N1-6
 BETRAG EINER GLEITKOMMAZAHN K2-5
 BEZUEGE B3-12,C2-1,5-3
 BITFELD G1-8
 -ANGABE G1-8
 -ENDE G1-8
 -INHALT G1-8
 -KONSTANTE G1-8

BITMUSTER H3-3
 BOOLESCHE OPERATIONEN N6-1
 BRINGEBEFEHL H3-1,K3-1,L4-1,9-1,10-1

 CODE C1-1,3-3
 -, EXTERN C1-1
 -, INTERN C1-1
 COMMONZONE B3-5,3-9,G2-7

 DATENBEREICH B3-3,G2-4
 DEHNUNGSWERT N4-2
 DEHNUNG DES SUCHVORGANGES N4-2,4-6
 DENORMALISIEREN K2-3
 DEZIMALE ANGABEN C4-1
 DEZIMALKOMMA G1-1
 DIFFERENZ
 -ANGABE C4-3
 -BILDUNG C3-5
 DISJUNKTION N6-2
 DIVISION H2-5,K3-7
 DOPPELCODEBEFEHL M9-1,10-7
 DOPPELTE WORTLAENGE H2-1,4-1
 DRITTELWORT C2-2,3-2,3-4,L4-2,5-2
 DRUCKPARAMETER G2-11

 EINERRUECKLAUF H2-4,K2-6
 EINFACHE WORTLAENGE H2-1,3-1,K3-1
 EINZELZEICHEN L9-4
 ENDE-BEFEHL G2-1
 ERSETZUNG D4-1,M9-1
 -, EINFACH D4-1
 -, VERSCHACHTELT D4-2
 ERSETZUNGS
 -BEFEHL M9-1,10-1
 -BEZUG D4-1
 -DEFINITION D4-1
 -PARAMETER D4-2
 -TEXT D4-1
 EXPONENTEN G1-1,K2-1
 -UEBERLAUF K2-4
 -UNTERLAUF K2-4
 EXTERNCODE C1-1

 FESTKOMMA H4-1
 -KONSTANTE MIT HALBWORTLAENGE G1-3
 -KONSTANTE MIT GANZWORTLAENGE G1-3
 -ZAHLEN H2-1,N1-2,1-3
 FORMALER PARAMETER D7-2
 FREIHALTE
 -ANWEISUNG C6-4
 -ZONE B3-5,3-9,G2-7
 FUELLBEFEHL N2-4

 GANZWORT C2-2,G1-4,L4-1,5-1,8-1
 -ADRESSE C2-1,1-1
 -LITERALE C7-1

IAS - Handbuch

April 69

GEBIETS
 -NAME B6-5
 -NUMMERN B6-5
 GELTUNGSBEREICH C6-5
 GLEICHSETZUNG G2-13
 GESTRECKTER SCHIFT N1-2
 - BEI NICHTZAHLWOERTERN N1-5
 GLEITKOMMA K2-1
 -EXPONENT N2-7
 -KONSTANTE EINFACHER GENAUIGKEIT G1-1
 -KONSTANTE DOPPELTER GENAUIGKEIT G1-2
 -ZAHLEN K2-1,N1-2
 GLOBALE
 - MAKRONAMEN B6-4
 - MAKROPARAMETER B6-4,D7-19
 - NAMEN B6-1
 GROSSEITE B3-2,3-13,N2-3
 GRUNDEINSTELLUNG C6-5

HALBWORT B5-4,C2-2,G1-5,H3-2,L8-2
 HALBE WORTLAENGE H2-1,5-1
 HAUPTPROGRAMM C6-7
 HEXADE L3-1

IDENTITAET N4-2
 IE-SPEZIFIKATION B5-1
 INDEX M1-1
 -ADRESSE C3-1,6-1,G2-9
 -ADRESSE BEI QUELLENPROGRAMMEN C6-9
 -BASIS D6-5,D3-14,G2-9
 -BASISREGISTER N6-3
 -BASISUMSCHALTUNG C6-6
 -BEFEHL C6-2,G2-9,M4-1
 -BEZUG C3-2
 -LISTE G2-10
 -NAMEN B6-1,C2-1,4-3
 -OPERATIONEN M4-1
 -PEGEL C6-3,2-10
 -SPEICHER M1-2
 -SPEICHERBEDARF D1-3
 -ZELLE D2-9,G2-9,M1-1,N2-5
 -ZONE C3-2,G2-9
 INDIREKTE SPRUENGE N2-2
 INDIREKTES SPRUNGZIEL N2-3
 INFORMATIONS
 -EINHEITEN B4-2,5-1,C2-3
 -TEIL B5-1
 INVERTIEREN L2-1

KERNESPEICHERBEZUG C3-5,2-1
 KOMMAVERSCHIEBUNG H2-5
 KOMMENTAR B5-5
 KONJUNKTION N6-1
 KONSTANTEN B5-2,G1-1
 -BEREICH B3-2,G1-1,2-4
 - IN BEFEHLSFORMAT G1-8
 KONTAKTNAMEN B6-2,G2-8
 KOPFWOERTER N4-6
 KREISSCHIFT L3-3,1-1,N1-2
 KURZSCHIFT N1-3
 - BEI NICHTZAHLWOERTERN N1-5
 - IM KREIS N1-5

LAGERKLASSE B3-10
 LANGSCHIFT N1-3
 - BEI NICHTZAHLWOERTERN N1-5
 - IM KREIS N1-5
 LEBENSDAUER DER GEBIETE B3-12
 LEERBEFEHL N1-1
 LINKS
 -ADRESSE C3-1
 -BUENDIGKEIT B5-4
 -SCHIFT H1-2,4-3,L7-1,N1-2

LITERALE C2-3,7-1
 - BEI ADRESSENBEFEHLEN C7-2
 - BEI BRINGEBEFEHLEN C7-2
 - BEI SPEICHERBEFEHLEN C7-3
 - BEI SPRUNGBEFEHLEN C7-2
 LOESCHEN H2-3,M2-1,N3-1
 - DER MARKE H2-3
 LOKALE NAMEN B6-1
 LUECKE B3-9
 LUECKENNAME B6-5

MAKRO D7-1
 -AUFRUF B5-2
 -NAME B6-1
 MAKROPARAMETER D7-2
 -, AKTUELLER D7-2,7-3
 -, FORMALER D7-2
 -, GLOBALER D7-19
 MAKROTEXT D7-1
 MANTISSE G1-1,K2-1
 MARKE H2-2,2-5,K2-6,N3-4
 - ALS ABRUCHKRITERIUM M11-5
 MARKEN
 -BIT H2-2
 -REGISTER H2-2
 MARKIERTES ZAHLWORT K2-6,H2-2
 MERKLICHTER M11-6,N2-6,3-5
 MODIFIKATOR M1-1,7-1,8-1
 MODIFIZIEREN M7-1,8-1
 MODIFIZIER
 -BEFEHL M7-1,8-1
 -GROESSE M7-1,8-1
 MONTAGE
 -CODE B4-1
 -NAME G2-8,B6-1
 -OBJEKT B4-1,G2-8
 -OBJEKTNAME G2-8
 -PROGRAMM B3-9,4-1
 MULTIPLIKATION H2-5,K3-6

NAMEN B5-2,6-1,7-2,C4-3
 NAMENLISTE B3-12
 NEGATION N6-3
 NICHTZAHLWOERTER L0-1
 NORMALISIEREN K2-3
 NULL N2-4
 -BEFEHL N2-4
 -FELD L10-1,N4-3

OKTADEN
 -DARSTELLUNG B7-1
 -FOLGEN B7-1,C4-2,G1-6
 -KONSTANTEN B1-5,G1-5
 -NAMEN B6-4,G1-7,2-11
 OPERAND C2-2
 OPERATIONSTEIL C1-1
 OPERATOREN B4-1,G2-12
 OPERATORLAUF B4-1

PARAMETER B3-10,C3-2
 -LISTEN D7-6
 PEGEL C6-3
 POSITIVE U. NEGATIVE NULL K2-3,H2-4
 PROGRAMM B4-1
 -ANFANG B4-2
 -ENDE B4-2
 -NAME B6-4
 -VERZWEIGUNG M1-3
 PROZESS-SPEZIFISCHER NAME B3-10
 PSEUDOBEFEHL B3-7,5-1,C6-2,G2-1

QUELLENBEREICH N5-1
 QUELLENBEREICHSADRESSE C3-6
 QUELLENPROGRAMM B4-1,6-1,C6-9,G2-1
 -AUFBAU G2-1

RECHENWERKSREGISTER N3-1
 RECHTS
 -ADRESSE C1-1,3-1
 -BUENDIGKEIT B5-4
 -SCHIFT H3-12,4-3,L7-1,N1-2
 REGISTER L3-1
 RELATIVBEZUG C5-2
 RELATIVE
 - ADRESSEN C5-2
 - SPRUENGE N2-2
 RESERVIEREN H3-1
 RUNDUNG H2-5,4-3,K2-4,N1-2,1-4
 - EINER FESTKOMMAZAHL H2-5
 - EINER GLEITKOMMAZAHL K2-4

SCHIFT H2-5,3-12,4-3,L7-1,M6-1,N1-1
 -BEFEHL L7-1
 -RICHTUNG L7-1,N1-2
 -ZAEHLER L7-2
 - BEI NICHTZAHLWOERTERN N1-1
 SCHLEIFENBILDUNG M0-1 11-1
 SCHRITTWEITE M11-2
 SEDIDEZIMALE B7-1,C3-6
 SEGMENTE B4-2,G2-1
 SEGMENTNAME B6-4
 SEITE B3-2,3-13
 SETZEN M2-1,N3-1
 - DER MARKE H2-3
 - EINER INDEXBASIS D1-5
 SICHERSTELLUNG N5-2
 SONDERZEICHEN B7-2,V5
 - ALS STEUERZEICHEN B7-2
 SPEICHER H3-3,L3-2
 -BEFEHL H3-3,3-5,K3-3,5-1,9-1,10-3
 -BEDARF D1-3
 -ZELLE N3-3
 SPEZIFIKATIONEN B5-2,5-3,C3-4,N1-1
 SPRUENGE N2-3
 -, ABSOLUTE N2-2
 -, BEDINGTE N2-2
 -, INDIREKTE N2-2
 - IN ANDERE GROSSEITE N2-3
 -, RELATIVE N2-2
 -, UNBEDINGTE N2-2
 SPRUNG
 -BEFEHL B3-3,L6-1,M1-3,N2-1
 -BEDINGUNG N2-1
 -ZIEL N2-3
 STANDARD-MAKROS D7-20
 STARTANWEISUNG G2-1
 STEUERPARAMETER V4
 SUBTRAKTION H2-5,K3-4,L8-1,10-2
 SUCH
 -VORGANG N4-2,4-6
 -WORT N4-4
 SUMMANDENADRESSEN C5-3
 SYMBOLISCHE
 -ADRESSIERUNG C5-1
 - ADRESSEN C5-1
 - INDEXADRESSEN C4-3,6-2
 - KERNSPEICHERADRESSEN C5-2,C5-1
 - NAMEN C5-1
 - PEGELADRESSEN C5-3
 SPEZIFIKATIONEN B5-3

TABELLENSUCHBEFEHL N3-4,4-1
 TABELLENDURCHSUCHEN N4-1
 - AUF GROESSER ODER GLEICH N4-4
 - AUF IDENTITAET N4-2
 - AUF MAXIMUM N4-7
 - AUF MINIMUM N4-7
 -, LOGARITHMISCH N4-4
 -, LOGARITHMISCH MIT RUNDUNG N4-6
 -, MIT DEHNUNG N4-2
 - MIT MASKE N4-3
 - MIT MASKE UND DEHNUNG N4-4
 TECHN.RUECKSPRUNGADRESSE M10-4,D2-1

TEIL-SEITEN G2-2
 TEILWOERTER H6-1,L4-3,5-3,10-1
 TETRADENFOLGEN B7-1,C4-1
 TETRADENKONSTANTEN B5-4,G1-4
 -, MIT GANZWORTLAENGE G1-4
 -, MIT HALBWORTLAENGE G1-5
 TEXT
 -FOLGEN C4-2
 -KONSTANTEN B5-4,G1-6
 TRANSPORT H4-1,5-1,M3-1
 TYPENKENNUNG B5-3,H2-4,K2-5,L2-1,3-2
 - ALS ABRUCHKRITERIUM M11-5
 TYPENKENNUNGS-ALARM H2-4

UEBERSCHRIFT B5-5
 UEBERSETZUNGSPROTOKOLL B5-5
 UMSCHALTEN DER INDEXBASIS D3-14,C6-6
 UMSCHLUESSELN L9-4
 UMSCHLUESSELTABELLE L9-5
 UMWANDL. FESTKOMMA IN GLEITKOMMA H3-12
 UNBEDINGTE SPRUENGE N2-2
 UNTERDRUECKUNG DES TAS-PROTOKOLLS G2-11
 UNTERPROGRAMM D1-1,2-1,3-1
 -SPRUNG D1-1,N2-4
 -STUFEN D1-1

VARIABLENBEREICH B3-2
 VERAENDERLICHENBEREICH G2-4
 VERARBEITUNGSKLASSE B3-10
 VERGLEICH K2-5,L2-2,6-1,N4-2
 - ALS ABRUCHKRITERIUM M11-4
 - BEI SPRUENGEN H2-6
 VERKUEZUNG H4-1,N1-5
 VERLAENGERUNG H4-1
 VERSIONEN D7-12
 VERSORGUNGS-BEFEHLE D2-9,3-10
 VERSORGUNGS-BLOCK D1-2,2-3,2-8,3-6
 VERSORGUNGS-PARAMETER D2-2,2-3,3-5
 VERZWEIGUNG N2-1
 VERZWEIGUNGSKRITERIUM H2-4
 VORBESETZUNG B3-10,D7-7
 VOREINSTELLUNG FUER OPERATOREN B4-1
 VORZEICHEN H2-1,K2-2,M1-2
 -ANGLEICHUNG H4-1

WIEDERHOLUNG D7-4,7-17
 WORTGRUPPENTRANSPORT N5-1
 WORTLAENGE
 -, EINFACHE H2-1,3-1,K3-1
 -, DOPPELTE H2-1,4-1
 -, HALBE H2-1,5-1

ZAEHLGROESSE M11-1
 ZAHLEN B7-1
 ZAHLWORT H2-2
 ZEICHEN B7-1,L4-2,5-2,9-1
 -ADRESSE L9-1
 -FOLGE B7-1
 -VORRAT L9-1
 ZIEL
 -ADRESSE C3-6
 -BEREICH N5-1
 -NAMEN B6-5
 ZULAESSIGER ZAHLENBEREICH H2-1
 ZUSAMMENSETZUNG N6-3
 ZWEITBEFEHL D3-6,M9-1
 ZWEITCODE C3-3,M9-1

TAS - Handbuch

April 69

TELEFUNKEN
COMPUTER

B

TAS - HANDBUCH

Aufbau



INHALT

B

<u>AUFBAU</u>	0 - 1
<u>1. DAS TEILNEHMER-BETRIEBSSYSTEM BS 3</u>	1 - 1
1.1. Funktioneller Aufbau	1 - 1
1.2. Systemkern	1 - 2
1.3. Systemakteure	1 - 2
1.4. Prozesse	1 - 3
1.5. Kontrollfunktion	1 - 3
1.6. Bearbeitung des Auftrags	1 - 4
1.7. Datenorganisation	1 - 5
<u>2. ÜBERBLICK ÜBER DAS PROGRAMMIERSYSTEM</u> ..	2 - 1
2.1. Operatoren	2 - 1
2.2. Kommandosprache	2 - 1
2.3. Übersetzungsvorgang	2 - 2
2.4. Mischung von Sprachen	2 - 3
2.5. Testhilfen	2 - 3
2.6. Dialogbetrieb	2 - 3
<u>3. GLIEDERUNG DES ADRESSENRAUMES</u>	3 - 1
3.1. Adressierungsbedingungen	3 - 2
3.2. Adressenzonen	3 - 4
3.3. Ablage der Informationseinheiten	3 - 6
3.4. Adressenzonen im Adressenraum	3 - 9
3.5. Großseite - Seite	3 - 13
<u>4. NIEDERSCHRIFT</u>	4 - 1
4.1. Voreinstellung für Operatoren	4 - 1
4.2. Programmanfang	4 - 2
4.3. Programmende	4 - 3
4.4. Segmente	4 - 3
4.5. Informationseinheiten	4 - 3
<u>5. INFORMATIONSEINHEITEN</u>	5 - 1
5.1. Befehle	5 - 1
5.2. Pseudobefehle	5 - 1
5.3. Konstanten	5 - 2
5.4. Benennung	5 - 2
5.5. Spezifikationen	5 - 3
5.6. Abgrenzungsteil	5 - 5
5.7. Kommentar	5 - 5
5.8. Überschrift	5 - 5
<u>6. GELTUNGSBEREICH VON NAMEN</u>	6 - 1
6.1. Lokale Namen	6 - 1
6.2. Globale Namen	6 - 1
6.3. Montagenamen	6 - 1
6.4. Kontaktnamen	6 - 2
6.5. Globale Makronamen	6 - 4
6.6. Globaler Makroparameter	6 - 4
6.7. Programmname	6 - 4
6.8. Segmentname	6 - 4
6.9. Oktadename	6 - 5
6.10. Zonenname	6 - 5
6.11. Gebietsnamen und Lückennamen	6 - 5



7.	<u>ELEMENTE DER TAS-SPRACHE</u>	7 - 1
7.1.	Zeichen	7 - 1
7.2.	Zahlen	7 - 1
7.3.	Zeichenfolge	7 - 1
7.4.	Namen	7 - 2
7.5.	Sonderzeichen als Steuerzeichen	7 - 2

B

Im Teilnehmer-Rechensystem TR 440 wird ein Benutzer-auftrag in einer Folge von Operatorläufen bearbeitet. Dabei haben sowohl das übersetzte und montierte Quel-lenprogramm als auch diejenigen Programme, die zur Übersetzung, Montage u.a. dienen, die Eigenschaften von Operatoren (s. 2.1).

In dieser Schrift wird gezeigt, wie Quellenprogramme in der maschinenorientierten Sprache TAS erstellt werden mit dem Ziel, den Aufbau eines Operators wesentlich zu beeinflussen.

Darüber hinaus erscheint es sinnvoll, dem TAS-Programmierer in knapper Form darzulegen, wie ein Benutzer-auftrag für das Teilnehmer-Rechensystem TR 440 formuliert wird, wie der Auftrag im Rahmen des Teilnehmer-Betriebssystems BS 3 bearbeitet wird und welche Stellung ein Operator innerhalb dieses Betriebssystems hat.

Im Abschnitt 1 wird zunächst ein Überblick über das Teilnehmer-Betriebssystem BS 3 gegeben. Weitere Einzelheiten hierzu finden sich in der Schrift: TR 440 Teil-nehmer-Betriebssystem BS 3 Einführung.

Abschnitt 2 zeigt kurz die Leistungen des Programmiersystems, insbesondere den Übersetzungsvorgang. Außerdem wird hier kurz die TR 440-Kommandosprache beschrieben, die u.a. zur Steuerung der Dienstleistungen des Program-miersystems dient. Weitere Einzelheiten hierzu finden sich in den Schriften: TR 440 Programmiersystem und TR 440 Unterlagensammlung Kommandosprache.

Im Abschnitt 3 ist die Gliederung des Adressenraums dar-gestellt. Sie gilt für jedes Quellenprogramm und für Ope-ratoren.

Im Abschnitt 4 ist aufgezeigt, was bei der Niederschrift eines Programms beachtet werden muß, und im Abschnitt 5 ist der Aufbau einer Informationseinheit, die das Grund-element eines Programms bildet, beschrieben.

Abschnitt 6 befaßt sich mit dem Geltungsbereich von Namen.

Im Abschnitt 7 sind die Elemente der TAS-Sprache zu-sammengefaßt.

Im Teilnehmer-Rechensystem TR 440 kann eine Vielzahl von (bis zu 72) Benutzeraufträgen gleichzeitig zur Bearbeitung anstehen. Unter diesen können max. 48 Gespräche sein (s. 1.6).

Das Teilnehmer-Betriebssystem BS 3 dient dazu, jeden dieser Aufträge gemäß seiner Wichtigkeit, seinen Anforderungen an Betriebsmittel (s. 1.5) und gemäß den aktuell verfügbaren Betriebsmitteln mit anderen Aufträgen zeitlich verschachtelt zu bearbeiten und dabei eine möglichst gute Ausnutzung aller Betriebsmittel zu erzielen.

1.1. Funktioneller Aufbau

Das Betriebssystem BS 3 ist durch einen modularen Aufbau gekennzeichnet und liegt in seinem residenten Teil sowohl im Kernspeicher des RD 441 als auch im Kernspeicher des als Satellitenrechner verwendeten RD 186. Der zuladbare Teil liegt (wie das gesamte Programmiersystem) auf dem Trommelspeicher.

Der Teil des BS 3, der im RD 441 liegt, gliedert sich in Systemkern und eine feste Anzahl von selbständigen Programmeinheiten, Akteure genannt. Die Aufgaben der Akteure lassen sich wie folgt klassifizieren:

- Aufgaben, die zeitlich absolut vrrangig sind:
Erledigung der Aufgaben durch die Systemakteure.

- Aufgaben, die wenig rechenintensiv und stark ein/ausgabeintensiv sind; sie verursachen lange Wartezeiten auf die - im Vergleich zum Rechnerkern - langsamen peripheren Einheiten; Erledigung der Aufgaben durch die Vermittlerprozesse.
- Aufgaben, die rechenintensiv sein können und deren Erledigung zeitlichen Bedingungen (Reaktionszeit an den Konsolen) unterliegt. Erledigung der Aufgaben durch die (Gesprächs-)Abwickler.
- Aufgaben, die rechenintensiv sein können und deren Erledigung keinen zeitlichen Bedingungen unterliegt. Erledigung der Aufgaben durch die (Abschnitts-)Abwickler.

Aufgrund der aufgeführten Aufgabenklassen ist eine feste Zuordnung zwischen Akteur und Priorität getroffen worden, siehe Bild 1.1. (In einer späteren Version werden die Prioritäten dynamisch vergeben.)

Die angegebene Reihenfolge der Aufgabenklassen, die sich in den Prioritäten (s. Bild 1.1) wiederholt, um der Forderung nach vernünftiger Antwortzeit und gutem Durchsatz, d.h. optimaler Anlagenauslastung zu genügen.

Die Menge der Akteure zerfällt in Systemakteure und Prozesse, die sich durch unterschiedliche Aufgabenstellung und durch ihren Adressierungsmodus (s. TR 440 Eigenschaften des RD 441) unterscheiden. Bild 1.1 ist nach Priorität und nach Adressierungsmodus gegliedert. Die Erläuterungen der verwendeten Abkürzungen folgen im Text.

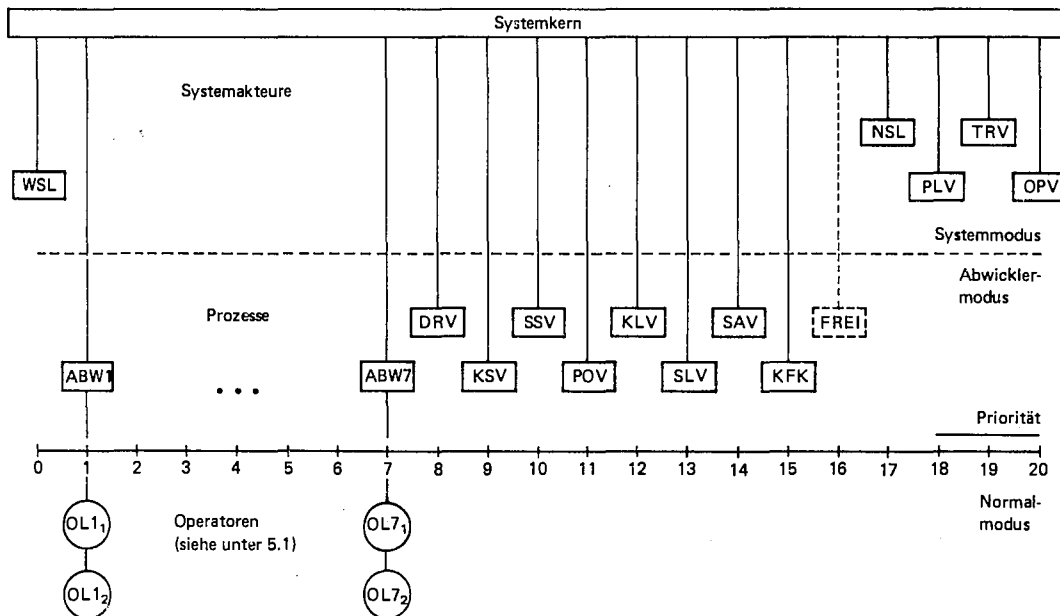


Bild 1.1 BS3 - Software - Organisation im RD 441

1.2. Systemkern

Der Systemkern verwaltet im BS 3 die Betriebsmittel Rechnerkern, Kernspeicher (im RD 441), Ein-/Ausgabegeräte und Kanäle.

Mit Verwaltung ist hierbei die "physikalische" Verwaltung (Vergabe und Rückgabe) der Betriebsmittel gemeint; die "logische" Verwaltung (Verplanen und Entplanen) führt die Kontrollfunktion durch (s. 1.5). Im einzelnen erfüllt der Systemkern folgende Aufgaben:

1.2.1. Rechnerkernvergabe

Dem Systemkern stehen selbständige Programmeinheiten - Akteure - gegenüber, die zueinander eine feste Priorität haben. Bedingt durch die Abhängigkeit der einzelnen Akteure voneinander, gibt es Situationen, in denen einzelne Akteure an der Zuteilung des Rechnerkerns uninteressiert sind, d.h. sie haben den Zustand "nicht rechenwillig".

Andere Akteure bewerben sich um den Rechnerkern, sie besitzen den Zustand "rechenwillig". Der Systemkern teilt dem rechenwilligen Akteur mit der höchsten Priorität den Rechnerkern zu.

1.2.2. Speicherverwaltung

Die Speichermedien (Kernspeicher und Hintergrundspeicher) werden in Einheiten von 1K (1024) Ganzworten verwaltet, der Kernspeicher vom Systemkern, die Hintergrundspeicher von Systemakteuren (s. 1.3). An diese Verwaltungen werden von den Prozessen Forderungen nach freiem Speicherraum und nach Beendigung ihrer Aufgaben deren Freigabe gemeldet.

1.2.3. Koordination der Peripheriegeräte

Der Systemkern übernimmt die Ein- und Ausgabebefehle von den Akteuren, verteilt diese auf Warteschlangen vor den EA-Kanälen und arbeitet sie über die Kanalwerke ab. Die Rückmeldungen von den EA-Kanälen werden den betroffenen Akteuren anschließend zugestellt.

Die Vermittlerprozesse melden bei Ein-/Ausgabeverkehr das betreffende periphere Gerät beim Systemkern an und nach Beendigung des Transports wieder ab.

In der Behandlung durch den Systemkern nimmt das "periphere Gerät" TR 86 S (Satellitenrechner RD 186 und dessen periphere Geräte, Fernschreiber und Sichtgerät) keine Sonderstellung ein. Ausgabebefehle an die Konsolen werden an das Kanalwerk geleitet, an dem der RD 186 angehängt ist.

1.3. Systemakteure

Die Systemakteure erfüllen systemnahe und hardwarenahe Aufgaben, die nachstehend stichwortartig beschrieben sind. Die Warteschleife WSL erhält immer dann den Rechnerkern, wenn kein Akteur den Zustand "rechenwillig" besitzt. Die Warteschleife hat zu diesem Zweck immer den Zustand "rechenwillig".

Die Notschleife NSL wird bei Systemfehlersituation rechenwillig gesetzt.

1.3.1. Plattenvermittler

Der Plattenvermittler PLV verwaltet den Plattenspeicher und organisiert den Informationstransport von und zu diesem Speichermedium.

1.3.2. Trommelvermittler

Für den Trommelvermittler TRV gilt dies sinngemäß für den Trommelspeicher.

1.3.2. Operateurvermittler

Der Operateurvermittler OPV belegt die höchste Priorität, um dem Operateur die Möglichkeit zu geben, jederzeit mit Hilfe von Operateurkommandos (z. B. zum Abfragen der Betriebsituation) in den Bearbeitungsablauf einzugreifen. Außerdem verwaltet der Operateurvermittler die Kontrollschreibmaschine und ermöglicht das Ein- und Ausgeben von Information zwischen den Akteuren und dem Operateur.

1.4. Prozesse

Unter Prozessen sind Programme zu verstehen, die für den Benutzer des Rechensystems genau definierte Leistungen erbringen. Während die Systemakteure ihre Aufträge von Prozessen erhalten und systemnahe Aufgaben erfüllen, also vom Benutzer nicht direkt erreichbar sind, stehen die Prozesse stellvertretend für den Benutzer dem Systemkern gegenüber.

Dabei werden Prozesse unterschieden, die die EA-Geräte bedienen - Vermittlerprozesse - und die Rechenaufträge des Benutzers verwalten - Abwicklerprozesse.

1.4.1. Vermittlerprozesse

Die Vermittlerprozesse DRV, KSV, SSV, POV, KLV und SLV (vgl. Bild 1.1) organisieren den Informationstransport zwischen Hintergrundspeichern und den peripheren Geräten Schnelldrucker, Kartenstanzer, Streifenstanzer, Plotter, Kartenleser und Streifenleser.

Dabei haben sie die Aufgabe, die Information in den Zentralcode oder aus diesem in den entsprechenden Gerätecode umzuschlüsseln und die entsprechenden Geräte zu betreiben.

Magnetbandgeräte werden durch einen Vermittler betrieben, der Bestandteil der Abwicklerprozesse ist. Aufgrund dieser Tatsache kann der Magnetbandverkehr - wie durch die Eigenheit des Magnetbands erforderlich - direkt betrieben werden.

1.4.2. Konsolsystem

Der Satellitenvermittler SAV ist gemeinsamer Kommunikationspartner aller mit Konsolen in Verbindung stehender Programmläufe. Er nimmt im Kernspeicher oder auf einem Hintergrundspeicher gepufferte Ausgaben entgegen und übergibt sie in Teilstücken nach Aufforderung an den RD 186.

Eingabeinformation von der Konsole nimmt der Satellitenvermittler vom RD 186 entgegen. Unvollständige Eingaben puffert er auf dem Hintergrundspeicher, vollständige Eingaben werden an die betreffenden Programmläufe weitergeleitet (siehe Bild 1.2).

Die Software im RD 186 wird ihrer Aufgabe gemäß Konsolverteilerprogramm (KVP) genannt. Dieses Programm organisiert die Konsolein- und -ausgabe und die Übertragung der Information vom und zum Satellitenvermittler über die Rechnerkopplung. Dabei besorgt es die Registrierung der verschiedenen Konsolzustände, puffert geringe Ein-/Ausgabemengen und sorgt für den geregelten Fluß der Information.

Die Programmkomplexe Satellitenvermittler SAV (im RD 441) und Konsolverteilerprogramm KVP (im RD 186) bilden zusammen das Konsolsystem KOS.

1.4.3. Abwicklerprozesse

Ein vom Benutzer gestellter Auftrag (Gespräch oder Abschnitt) hat den Ablauf einer Operatorlauf-Kette (Entschlüsselungslauf, Übersetzungslauf usw.) zur Folge. Das vom Benutzer eingegebene Programm wird zu einem Glied - dem Objektlauf - in dieser Kette (s. Abschnitt 2).

Ein Abwicklerprozeß hat die Aufgabe, diese Operatorlaufkette zu verwalten und den einzelnen Operatorläufen Systemdienste anzubieten. Aus der Sicht eines Operatorlaufs stellt der Abwicklerprozeß ABW ein benutzerspezifisches Betriebssystem dar.

1.5. Kontrollfunktion

Die Bearbeitung eines Benutzerauftrags beginnt normalerweise mit der Entgegennahme des Quellprogramms und endet mit dem Ausgeben der Ergebnisse auf den gewünschten Ausgabemedien. Dazwischen liegt die Bearbeitung des Auftrags durch die Prozesse, zu der "Betriebsmittel" benötigt werden. Unter diesem Begriff werden alle Hardware-Einrichtungen, Programme und Informationen zusammengefaßt, deren Verfügbarkeit nicht zu jedem Zeitpunkt gewährleistet ist, von denen es also zu wenig gibt, um in jedem Fall alle Interessenten gleichzeitig bedienen zu können.

1.5.1. Betriebsmittel

Die wesentlichste Aufgabe des Prozesses Kontrollfunktion KFK ist es, über die Vergabe der Betriebsmittel zu entscheiden. Betriebsmittel sind in diesem Zusammenhang:

- Rechnerkern
- Speichermedien (Kern- und Hintergrundspeicher)
- Abwicklerprozesse
- Ein- und Ausgabegeräte (über Vermittlerprozesse)
- Auftrags-elemente (Kernspeicherbereiche zur internen Beschreibung eines Auftrags)

Der Satellitenvermittler nimmt das XBG-Kommando entgegen und teilt der Kontrollfunktion mit, daß ein Abwicklerprozeß den Auftrag übernehmen muß. Es entsteht ein Wechselgespräch zwischen dem Benutzer und den von ihm angesprochenen Operator, die von dem Abwickler verwaltet werden. Die Information wird dabei je nach Umfang im Kernspeicher oder über den Plattenspeicher ausgetauscht. Zusätzlich kann der Benutzer auf die Datenbestände in der langfristigen Datenhaltung LFD zugreifen (vgl. 1.7). Nach Gesprächsende wird der Druckervermittler beauftragt, die auf dem Plattenspeicher angefallene Druckinformation auszugeben.

1.6.1. Verdrängen von Gesprächen

Der Grundgedanke eines Teilnehmer-Rechensystems ist die zeitliche Aufteilung (time sharing) der begrenzt zur Verfügung stehenden Betriebsmittel an die Programme der einzelnen Benutzer.

Da der Mensch eine gegenüber dem Rechner große Reaktionszeit besitzt, merkt er in den meisten Fällen von der Zeiteilung nichts. Er hat den Eindruck, die Leistung der Rechenanlage stehe ihm allein zur Verfügung.

Jedes Gespräch darf seit der letzten Zuteilung eines Abwicklerprozesses ein bestimmtes Zeitintervall - Bedienungsintervall - im Besitz des Rechnerkerns sein.

Gibt ein Gespräch innerhalb des Bedienungsintervalls nicht von sich aus einen Anlaß zur Verdrängung - Konsolausgabe mit anschließender Eingabe - so wird der Programmbestand des Gesprächs zwangsweise auf den Trommelspeicher verdrängt (swapping). Die frei gewordenen Betriebsmittel stehen anderen Gesprächen zur Verfügung.

1.7. Datenorganisation

Die Datenorganisation bildet als Teil der Abwicklerprozesse einen abgeschlossenen Programmkomplex. Zur Speicherung von Datenbeständen stehen dem Benutzer des Teilnehmer-Rechensystems TR 400 die Speichermedien Trommel, Platte und Magnetband zur Verfügung. Zum Erstellen, zur Verwaltung und zur Verarbeitung dieser Datenbestände werden ihm die Dienstleistungen der Datenorganisation angeboten.

1.7.1. Dateien

Der Benutzer (und z.T. auch das Betriebssystem) hält seine Information in Form von Dateien. Der Begriff Datei beinhaltet, daß die Information strukturiert ist und daß über die Datei Verweisinformation gehalten wird. Die Dateien werden aus Sätzen - den kleinsten adressierbaren Informationseinheiten - aufgebaut. Nach der Art des Zugriffs auf diese Sätze werden drei Dateitypen unterschieden:

- SEQ Sequentieller Zugriff auf die Sätze
Die physikalische Reihenfolge der Sätze auf dem Speichermedium bestimmt die logische Folge.
- RAN Randomzugriff mit Satznummern
Die Information liegt dicht gepackt in der Reihenfolge ihres Eintreffens. Die logische Reihenfolge wird durch Stellvertreter repräsentiert, die nach ihrem Binärwert lückenlos aufsteigend angelegt sind.
- RAM Randomzugriff mit Satzmarken
Der Unterschied zu RAN liegt darin, daß die Variationsmöglichkeit der Stellvertreter bei gleichem Speicherbedarf ungleich höher ist, da für undefinierte Sätze auch kein Stellvertreter angelegt wird.

Innerhalb eines Auftrages können simultan bis zu 255 Dateien verwaltet werden. Der Benutzer hat die Möglichkeit, die Dateien logisch zusammenzufassen unter dem Begriff einer Datenbasis. Der Dateiname braucht dann nur innerhalb einer Datenbasis eindeutig zu sein.

Die Information einer Datei kann der Benutzer nach Belieben handhaben, insbesondere kann die Datei für Zugriffe gesperrt werden.

1.7.2. Bibliotheken

Die Zahl der Datenbasen, die ein Abwicklerprozeß verwalten kann, ist auf 8 begrenzt.

Zwei ausgezeichnete Datenbasen werden durch das Betriebssystem aufgebaut und bearbeitet:

- Eine öffentliche Datenbasis (Systembibliothek)
Die öffentliche Datenbasis beinhaltet den Programmbestand des Programmiersystems. Sie steht allen Benutzern für lesenden Zugriff zur Verfügung. Die öffentliche Datenbasis wird bei jedem Systemaufbau neu aufgebaut, ihre Lebensdauer stimmt also mit der des Systems überein.
- Eine Standarddatenbasis (Auftragsbibliothek)
Jedem Benutzerauftrag steht eine Standarddatenbasis zur Verfügung, die auftragspezifische Daten enthält (z.B. Montageobjekte, Rückverfolgungslisten, Operatorkörper-Beschreibung).

Die Standarddatenbasis wird von einem Operator - dem Entschlüsselner (s. unter 2) - aufgebaut und mit Systemdateien besetzt. Auch der Benutzer legt implizit seine Dateien in die Standarddatenbasis, wenn er keine eigene Datenbasis (s.o.) definiert.

Die Standarddatenbasis wird am Ende eines Benutzerauftrags gelöscht.

1.7.3. Langfristige Datenhaltung (LFD)

Die langfristige Datenhaltung bietet auch dem Benutzer die Möglichkeit, Dateien permanent zu speichern. Dabei werden die Dateien des Benutzers unter einem gemeinsamen Benutzerkennzeichen verwaltet, das an die Stelle eines Datenbasisnamens tritt (Benutzerbibliothek). Die Entscheidung über den Inhalt und die Lebensdauer der Dateien in der LFD liegt beim Benutzer, ähnlich wie die Rechenzentrumsleitung über die Maintenance den Inhalt der öffentlichen Datenbasis festlegen kann.

Die Dateien in der LFD - alle oben genannten Typen sind zugelassen - unterscheiden sich in ihrer Zugriffseigenschaft:

- G-Datei (Gemeinschaftsdatei), die für den Zugriff mehrerer Benutzer bestimmt ist und für die zwangsweise eine Koordinierung simultaner Bearbeitung stattfindet.
- P-Datei (Privatdatei), die nur von dem Benutzer veränderbar und löschar ist, der diese Datei kreiert hat.

Manipulationen an Dateien der langfristigen Datenhaltung sind durch Kommandos der Texthaltung (s. 2.6) und von höheren Programmiersprachen her möglich.

1.7.4. Gebietskonzept

Die Notwendigkeit der Verlagerbarkeit von größeren Informationseinheiten zwischen verschiedenen Speichermedien - z.B. Verdrängung eines Programmbestandes - führt zur indirekten Datenadressierung.

Zusammenhängende Informationseinheiten von einer oder mehreren Seiten (je IK TR 440 -Wörter) werden als Gebiet bezeichnet.

Auf in Gebieten organisierte Informationsbestände kann der Benutzer implizit (über die Datenorganisation) oder explizit (Dienstleistungen des Abwicklers) zugreifen.

Die Gesamtheit aller Operatoren, die jedem Benutzer des Teilnehmer-Rechensystems TR 440 zur Verfügung stehen - Standardoperatoren - und aller Prozeduren, die als Montageobjekte gehalten werden, wird als Programmiersystem bezeichnet.

2.1. Operatoren

Ein Operator im Sinne der Software-Organisation des TR 440 ist jedes Programm, das folgenden Bedingungen genügt:

- alle Befehle laufen im Normalmodus ab (s. TR 440 Eigenschaften des RD 441)
- alle Beziehungen zum Betriebssystem werden über einen Abwicklerprozeß hergestellt, der den zugehörigen Benutzerauftrag bearbeitet.
- im Abwickler existiert eine Beschreibung dieses Programms (Startadresse, Größe usw.).

Aufgrund der zuletzt genannten Beschreibung ist der Abwickler in der Lage, einen Operator zu starten, d.h. einen Operatorlauf aus ihm zu machen.

2.1.1. Montierer

Auch das Programm des Benutzers wird zu einem Operatorlauf in der Kette der Bearbeitungen - dem Objektlauf.

Es ist natürlich nicht Aufgabe des Benutzers, die dazu notwendige Beschreibung zu erstellen. Diese Aufgabe übernimmt ein spezieller Operator - der Montierer.

Er baut aus einzelnen Montageobjekten (vgl. 2.3) ein lauffähiges Programm auf und hinterlegt die Beschreibung in der Standarddatenbasis des Benutzers (vgl. 1.7). Der Montagevorgang kann speziell vom TAS-Programmierer nach Wunsch gesteuert werden (s. 3.4).

Zu der Beschreibung gehören auf Wunsch des Benutzers auch Referenzlisten, die es den zahlreichen Analyse-Operatoren (s. 2.5) erlauben, im Fehlerfall des Programmablaufs Informationen mit den Bezeichnungen in der Quelle des Benutzers auszugeben (quellenbezogener Dump) und die Fehlerstelle zu lokalisieren.

2.1.2. Entschlüssler

Ein Benutzerauftrag besteht aus der eigentlichen Information, die von der Rechenanlage zu verarbeiten ist (Quelle, Daten) und Angaben darüber, in welcher Form das geschehen soll. Die Steuerinformation in Form von Kommandos (s.u.) wird von einem speziellen Operator, dem (Programmiersystem-) Entschlüssler, ausgewertet. Der Entschlüssler veranlaßt daraufhin den Start der geforderten Operatoren.

2.2. Kommandosprache

Um dem Benutzer ein einheitliches und bequemes Steuermittel für den Bearbeitungsablauf in die Hand zu geben, wurde die TR 440-Kommandosprache geschaffen. Ihre Gültigkeit beschränkt sich nicht nur auf das Programmiersystem, vielmehr läßt sich mit ihr die gesamte Externsteuerung des Rechensystems formulieren.

Die Elemente der Sprache sind Kommandos, die mit einem speziellen Zeichen, dem Fluchtsymbol (im weiteren durch \diamond dargestellt), beginnen müssen, um sie von den übrigen Eingabedaten trennen zu können.

2.2.1. Tätigkeitskommandos

Das Grundelement der Kommandosprache ist das Tätigkeitskommando, mit dem eine bestimmte Leistung vom Programmiersystem verlangt wird. Es beginnt mit dem Tätigkeitsnamen, auf den Spezifikationen folgen, z.B.:

\diamond UEBERSETZE, SPRACHE = FTN, QUELLE = TEXT, ...

Für jede Tätigkeit ist eine Menge von Spezifikationen definiert (im Beispiel SPRACHE, QUELLE usw.). Hält man eine vorgegebene Reihenfolge ein, können die Spezifikationsnamen entfallen. Auch das Abkürzen der Tätigkeits- und Spezifikationsnamen ist möglich, solange Eindeutigkeit gewährleistet ist, z.B.:

\diamond UEB., FTN, QU. = TEXT, ...

2.2.2. Voreinstellung

Zu den einzelnen Spezifikationen, die in obligate und optionale unterteilt werden, existiert eine globale Voreinstellung.

Diese Voreinstellung tritt in Kraft, wenn zu einer Spezifikation keine Aussage gemacht wird. Zu den Tätigkeitskommandos zählen:

- Grunddienste
Übersetzen, Montieren, Starten und Löschen von Objekten
- Allgemeine Dateidienste
Einrichten, Einschleusen, Sichern und Sperren von Dateien
- Texthaltungsdienste
Eintragen, Korrigieren, Mischen und Kopieren von Texten
- Datenmanipulationen
Sortieren und Mischen von Sätzen, Komprimieren von Quellen
- Datentransportdienste
Ein- und Ausgabe von Binärobjekten, Ausgabe von Daten
- Assembler-Makrodienste
Eintragen, Löschen und Informieren

- Langfristige Datenhaltung
Einrichten, Aufgeben und Verwalten von langlebigen Daten

2.2.3. Erweiterung des Kommandovorrats

Der vorhandene Kommandovorrat läßt sich über das DEFINIERE-Kommando oder über eine Prozedurvereinbarung erweitern, ohne daß dabei bestehende Kommandos versehentlich gelöscht werden können.

Durch das DEFINIERE-Kommando neu geschaffene Kommandos starten und versorgen bei Aufruf den bei der Definition angegebenen Operator, den der Benutzer selbst geschrieben hat.

Durch eine Prozedurvereinbarung wird eine Folge von vorhandenen Kommandos zu einem neuen Kommando zusammengefaßt. Dabei können formale Parameter eingeführt werden, die dann als Spezifikationen des neuen Kommandos auftreten. Beispiel:

Prozedurvereinbarung:

- ◇ *RECHNE (TEXT, SPRACHE, PROG)
- ◇ UEBERS., QUELLE = *TEXT, SPR. = *SPRACHE
- ◇ MONTIERE, PROGRAMM = *PROG
- ◇ STARTE, PROGR. = *PROG, DUMP = F-NEST'A-NEST
- ◇ * *

Prozeduraufruf:

- ◇ RECHNE, TEXT = QUELLE, SPR. = FTN, PROG = TEST

2.2.4. Kommandoablauf

Wird der Auftrag des Benutzers im Abschnittsmodus bearbeitet, so kann die Übersetzung einer Quelle fehlerhaft verlaufen, ohne daß der Benutzer wie bei einem Gespräch dies feststellen kann. Ein nachfolgender Montage- und Startversuch wäre sinnlos. Für solche Fälle gibt es zwei spezielle Kommandos, die den Auftrag nach einem aufgetretenen Fehler abbrechen (FEHLERHALT) oder in Abhängigkeit von auftragspezifischen Variablen die vorgegebene Kommandofolge verlassen (SPRINGE).

2.3. Übersetzungsvorgang

Eine der wichtigsten Leistungen des Programmiersystem ist das Übersetzen einer Quelle in die darin gewünschte Rechnerleistung. Die hierzu erforderliche Übersetzung erfolgt grundsätzlich zweistufig (s. Bild 2.1).

Die Quelle, die entweder in einer Datei oder in der Form von Eingabedaten auf einem Hintergrundspeicher liegt, wird zunächst von einem Sprachübersetzer verarbeitet. Dieser erzeugt daraus ein Montageobjekt, das aus dem entsprechenden Montagecode und aus Adreßbüchern für die Rückverfolgung im Fehlerfall besteht, falls Dumpfähigkeit verlangt wird (Spezifikation im Kommando). Der Montagecode selbst stellt keinen lauffähigen Maschinencode dar. Dieser wird erst bei der sich anschließenden Montage erzeugt.

Der Montierer sorgt für die Verknüpfung und günstige Anordnung von einem oder mehreren Montageobjekten in dem von ihm daraus erstellten lauffähigen Operator (vgl. Bild 2.1).

Im Bild 2.1 ist der Übersetzungsvorgang dargestellt, wie er beispielsweise verlaufen kann, solange der Abwicklerprozess den Auftrag bearbeitet (vgl. Bild 1.2).

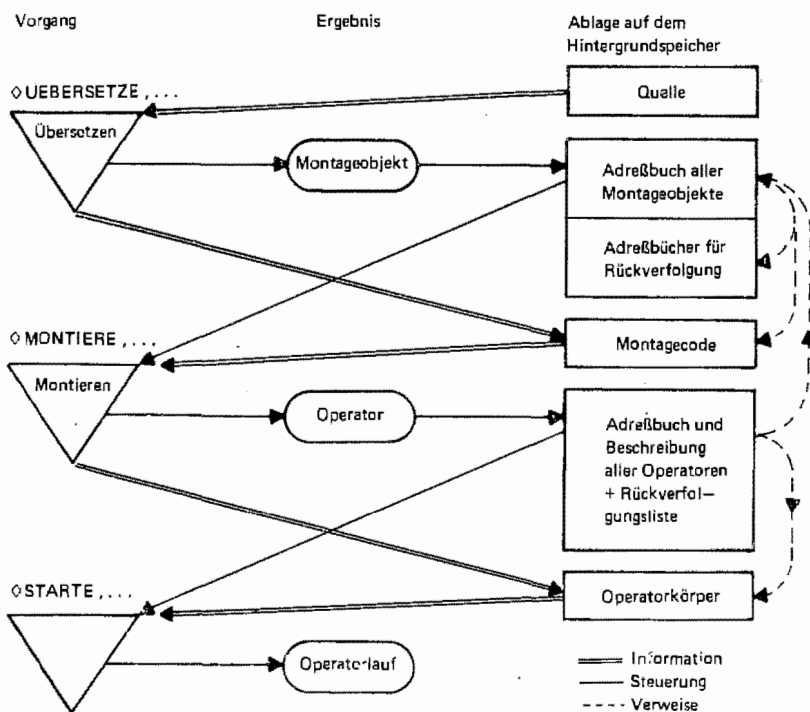


Bild 2.1 Übersetzungsvorgang einer Quelle und Start des erzeugten Operators

2.4. Mischung von Sprachen

Die Zweistufigkeit der Übersetzung dient u.a. dazu, Prozeduren aus verschiedenen Sprachen gemeinsam verwenden zu können. Dabei ist der gegenseitige Aufruf in den Sprachen ALGOL60, FORTRAN und TAS realisiert. Aus TAS heraus können auch COBOL-Routinen, aus COBOL Routinen der restlichen genannten Sprachen aufgerufen werden.

Der Zusammenschluß erfolgt während des Montagevorgangs. Der Übergang zwischen zwei Sprachen innerhalb einer Prozedur (z.B. Einfügen von TAS-Befehlen in ein ALGOL-Programm) ist nicht möglich.

Für den Anschluß von TAS-Prozeduren an höhere Sprachen sind nur die Konventionen einzuhalten, nach denen ein bestimmter Compiler seine Montageobjekte erzeugt (s. Kapitel D, Abschnitte 8 und 9).

Ebenso sind beim Anruf von Prozeduren höherer Sprachen aus einem TAS-Programm die Konventionen der jeweiligen Sprache zu berücksichtigen.

2.5.3. Tracing

Eine weitere Testhilfe besteht in der Möglichkeit, ein Programm überwacht ausführen zu lassen. Der Ausdruck von Registerständen oder Speicherbereichen kann dabei vom Ansprechen eines bestimmten Speicherbereichs oder vom Auftreten eines bestimmten Befehlscodes abhängig gemacht werden (z.B. Protokollierung aller auftretenden Sprungbefehle, s. Kapitel E, Abschnitt 7).

Ähnliche Ablaufüberwachungen sind auch bei Programmen möglich, die aus Quellen in höheren Programmiersprachen entstanden sind. Die Routinen werden im UEBERSETZE-Kommando angefordert und im STARTE-Kommando aktiviert.

2.5.4. Dynamische Kontrollen

(für höhere Programmiersprachen)

Auf Wunsch des Benutzers werden beim Übersetzungsvorgang (TAS-Quellen ausgenommen!) dynamische Kontrollen eincompiliert, die sich beziehen auf:

- Einhaltung von Indexgrenzen
- Verträglichkeit von aktuellen und formalen Parametern
- Zulässigkeit von Schleifenparametern

Damit lassen sich statisch und syntaktisch nicht erkennbare Fehler lokalisieren.

2.5. Testhilfen

In diesem Abschnitt sollen die Hilfen beschrieben werden, die das Programmiersystem zum Austesten von Programmen zur Verfügung stellt.

2.5.1. Quellbezogene Dumps

Hierzu gehören in erster Linie quellbezogene Dumps, die bei irregulären Abbrüchen der Bearbeitung oder auch auf explizites Verlangen des Benutzers die Variablen ausdrucken. Dabei werden die in der Quelle angegebenen Namen von Variablen und deren Inhalt in lesbarer Form ausgedruckt, so daß Referenzlisten nicht benutzt werden müssen.

Die Druckausgabe des Dumps kann weitgehend gesteuert werden. So kann u.a. der Ausdruck aller Variablen mit einzelnen Ausnahmen (z.B. große Felder) oder nur der Variablen, die an der aktuellen Aufrufverschachtelung beteiligt sind oder der Druck explizit angegebener Variablen verlangt werden.

2.5.2. Rückverfolger

Von einem ausgezeichneten Operator - dem Rückverfolger - wird außerdem die Fehlerstelle, ebenfalls unter Bezugnahme auf die Originalquelle, lokalisiert. Zusätzlich wird die aktuelle Aufrufverschachtelung ausgegeben. Welche Dumps in welcher Form im Alarmfall ausgeführt werden sollen, wird im STARTE-Kommando spezifiziert (s. auch Kapitel E, Abschnitt 4).

2.6. Dialogbetrieb

Im Gespräch von einer Konsole stehen bis auf unwesentliche, durch den Gesprächsmodus bedingte Einschränkungen alle Dienstleistungen des Programmiersystems zur Verfügung.

Nach einer Eröffnung eines Gesprächs besteht die Möglichkeit, ein einzelnes oder eine Folge von Kommandos einzugeben.

Im Gesprächsmodus können dabei in eine bereits eingegebene Kommandofolge Kommandos eingefügt werden, die dann vorrangig ausgeführt werden.

Beispielsweise läßt sich bei einer fehlerhaften Übersetzung der Fehler mit einem Kommando korrigieren, um anschließend die Übersetzung zu wiederholen. Nach den vorrangig ausgeführten Kommandos wird nur auf expliziten Wunsch des Benutzers an der Unterbrechnungsstelle fortgefahren.

2.6.1. Anweisungen

Im Gesprächsmodus können außer Kommandos auch Anweisungen an den Abwicklerprozeß, der den Auftrag bearbeitet oder den Entschlüßler und einen gesprächsfähigen Objektlauf des Benutzers gegeben werden. Anweisungen werden immer nur einzeln und auf Anforderung verarbeitet. Der Entschlüßler versteht z. B. Anweisungen zum Fortsetzen an der Unterbrechungsstelle und zum Löschen aller eingegebenen und noch nicht ausgeführten Kommandos. Im Gegensatz zu FORTRAN- und ALGOL-Programmen ist die Gesprächsfähigkeit von TAS-Programmen nur durch expliziten Anschluß der Kontrollprozedur zu erreichen (vgl. Kapitel E, Abschnitt 4.5).

2.6.2. Texthaltung

Arbeitet der Benutzer im Gespräch mit umfangreichen Quellen und Datenmengen, so ist es unbequem, diese von der Konsole aus einzugeben. Vielmehr ist in diesem Fall der Gebrauch der langfristigen Datenhaltung zu empfehlen (vgl. 1.7).

Unabhängig von oder auch im Zusammenhang mit LFD-Dateien arbeitet die Texthaltung von Quellen. Mit ihrer Hilfe lassen sich Quellen in Dateien eintragen, einzelne Zeilen korrigieren, löschen oder auch vertauschen. Auch einzelne Zeichen einer Textzeile lassen sich eintragen, auswechseln oder löschen. Ferner kann der Benutzer mehrere Quellen zu einem Quellentext zusammenfügen, mischen oder Teile korrigieren.

2.6.3. Erweiterte Testmöglichkeiten

(für höhere Programmiersprachen)

Das Austesten eines Programms wird gerade im Gesprächsmodus durch die Möglichkeit, auf Ergebnisse sofort zu reagieren, vereinfacht und beschleunigt. Ein gesprächsfähig montierter Operator kann jederzeit während seines Laufs angehalten werden. Im UEBERSETZE-Kommando werden sogenannte Kontrollereignisse definiert, indem den Nummern von Quellzeilen Namen zugeordnet werden (TAS-Quellen ausgenommen!).

Ist ein Kontrollereignis aktiv (Angabe im STARTE-Kommando), so hält der Operatorlauf an der entsprechenden Stelle an und meldet den Namen des Ereignisses auf der Konsole. Darauf sind mehrere Reaktionen möglich:

- Fortsetzen oder Beenden des Operatorlaufs
- Aktivieren oder Passivieren von definierten Kontrollereignissen
- Bringen oder Umsetzen der Werte einzelner Variablen, die frei aus dem Programm wählbar sind
- Dump aller Variablen nach Anweisung auf die Konsole oder den Schnelldrucker (gepuffert)
- Einfügen von Kommandos aller Art
- Anhalten bei Operatorlaufende, d.h. vor Ausführung des nächsten anstehenden Kommandos.

3. GLIEDERUNG DES ADRESSENRAUMES

Der Adressenraum eines Operators umfaßt alle Befehle, Konstanten und die durch Befehle adressierbaren Daten. Daten, die sich außerhalb des Adressenraumes befinden, sind nicht durch Befehle adressierbar und können nur in Form von Gebieten vorliegen. Damit diese Daten adressierbar werden, müssen sie in den Adressenraum gebracht werden.

Bei der Assemblierung werden die einzelnen Informationseinheiten bestimmten Adressenzonen zugeordnet. Implizit sind 13 Adressenzonen vorgesehen, die bei Bedarf vom Assembler automatisch eingerichtet werden.

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

Darüber hinaus können explizit weitere Adressenzonen mit Hilfe der Pseudobefehle ZONE, FZONE und CZONE eingerichtet werden (siehe Abschnitt 3.2.2). Jede dieser Adressenzonen beginnt mit der Adresse 0. Bei der Montage werden die Adressenzonen durch Addition einer Translationsgröße im Adressenraum angeordnet. Nach der Montage sind die Adressenzonen vergessen.

Jede Adressenzone unterliegt bestimmten Adressierungsbedingungen und wird damit einem der Adressierungsbereiche zugeordnet. Die Adressierungsbereiche bilden damit eine Zusammenfassung der Adressenzonen, die gleiche Adressierungsbedingungen haben (siehe Abschnitt 3.1.).

Die Zuordnung der Informationseinheiten zu den verschiedenen Adressenzonen ist implizit vorgegeben. Wird eine andere Zuordnung gewünscht, so kann dies explizit angegeben werden (siehe Abschnitt 3.3.).

Für die Verlagerung eines Programms oder Teile davon innerhalb der Speicherhierarchie der Anlage werden die Adressenzonen zu Gebieten zusammengefaßt. Die Adressenzonen eines Gebietes werden von dem System als Einheit betrachtet. Die Zuordnung der Adressenzonen zu Gebieten wird vom Montageprogramm vorgenommen und kann durch explizite Angaben beeinflusst werden (siehe Abschnitt 3.4.).

Des weiteren wird auch die Anordnung der Gebiete im Adressenraum des Operators vom Montageprogramm vorgenommen. Diese Anordnung kann ebenfalls explizit beeinflusst werden (siehe Abschnitt 3.5.).

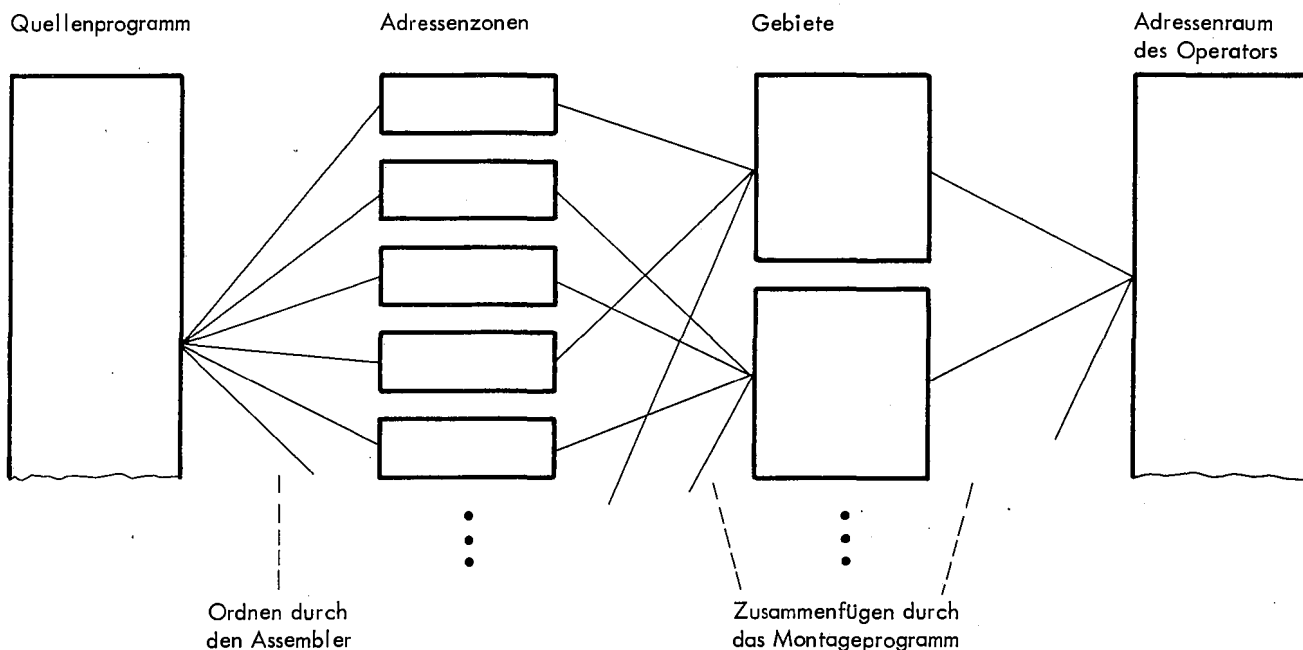


Bild 3.1 Ordnen der Informationseinheiten und Zusammenfügen

TAS - Handbuch

JAF. 69

3.1. Adressierungsbedingungen

Die Informationseinheiten eines Programms unterliegen gewissen Adressierungsbedingungen. Sie ergeben sich aus den Bedingungen

- Mit oder ohne Schreibschutz
- Adressierung mit 16-Bit-Adressen oder mit 22-Bit-Adressen
- Großseitengrenze darf überschritten werden oder nicht

Der Schreibschutz bezieht sich immer auf eine Einheit von 1024 Ganzwörtern bzw. 2048 Halbwörtern. Diese Anzahl von Wörtern wird als Seite bezeichnet. Es können also nur eine odere mehrere Seiten schreibgeschützt werden. Bei der ersten Adresse einer Seite sind stets die rechten 11 Bits Null.

Der Adressenteil von TR 440-Befehlen ist 16 Bits lang. Damit ist es möglich, maximal 32 768 Ganzwörter bzw. 65 536 Halbwörter zu adressieren. Diese Anzahl Wörter wird als Großseite bezeichnet. Sie umfaßt damit 32 Seiten. Bei der ersten Adresse einer Großseite sind die rechten 16 Bits stets Null. Die erste Großseite ist die Großseite Nummer 0.

Mit einer Adresse, die im Adressenteil eines TR 440-Befehls steht, können nur die Speicherzellen der Großseite 0 adressiert werden. In Verbindung mit den Modifizier- und Ersetzbefehlen ist es möglich, mit 22 Bits langen Adressen zu arbeiten. Damit ist es dann möglich, die Speicherzellen aller Großseiten von 0 bis 63 zu adressieren.

Die Adressen des größten Teils der Sprungbefehle sind relativ zur Großseite und können auch nur Adressen innerhalb einer Großseite ansprechen. Es ist also mit diesen Sprungbefehlen nur ein Sprung innerhalb einer Großseite möglich. Damit ist es möglich, die Befehlsfolge in jede Großseite zu legen und trotzdem mit 16-Bit-Adressen zu arbeiten. Dadurch kann bei einem Speicherbedarf von maximal 32 Seiten für Befehle und maximal 32 Seiten für Daten nur mit 16-Bit-Adressen gearbeitet werden.

Übersteigt die Anzahl der Befehle die Grenze einer Großseite, so können die Befehle auf mehrere Großseiten verteilt werden. Um in eine andere Großseite zu springen, stehen Befehle zur Verfügung, bei denen ein "Sprung in eine Großseite möglich" ist.

Adressenlänge	Großseite	Schreibsperre	
		konstant	variabel
16 Bits	innerhalb Großseite 0	K	V
	beliebig	nicht möglich	
22 Bits	innerhalb einer Großseite	B (F)	BV (FV)
	beliebig	DK	D

Bild 3.2 Adressierungsbereiche

Aus den oben angeführten Bedingungen ergeben sich 8 Adressierungsbedingungen. Adressenzonen, die die gleichen Adressierungsbedingungen haben, fassen wir zum Adressierungsbereich zusammen. Werden nur die implizit vorgegebenen Adressenzonen verwendet, ergeben sich maximal 4 Adressierungsbereiche:

Konstantenbereich	(K-Bereich)
Variablenbereich	(V-Bereich)
Befehlsbereich	(B-Bereich)
Datenbereich	(D-Bereich)

Jede Informationseinheit hat also implizit eine der vorstehenden Adressierungsbedingungen. Durch Angabe einer Spezifikation hinter der Informationseinheit (siehe Abschnitt 5.5.) kann ihr eine andere der vorstehend angeführten Adressierungsbedingungen zugeordnet werden. Weitere Möglichkeiten sind im Abschnitt 3.3. beschrieben.

Werden mit Hilfe der Pseudobefehle ZONE, FZONE und CZONE explizit weitere Adressenzonen definiert, so können weitere Adressierungsbedingungen angegeben werden. Sie bilden die Adressierungsbereiche

Befehlsbereich variabel	(BV-Bereich)
Datenbereich konstant	(DK-Bereich)
Freier Befehlsbereich	(F-Bereich)
Freier Befehlsbereich variabel	(FV-Bereich)
Indexbereich	(I-Bereich)

Nachstehend sind die Adressierungsbedingungen dieser Adressierungsbereiche näher erläutert.

3.1.1. Konstantenbereich

Der Konstantenbereich ist schreibgeschützt und kann mit 16-Bit-Adressen adressiert werden. Er liegt stets in der Großseite 0. Ebenfalls in der Großseite 0 liegt der Variablenbereich. K- und V-Bereich zusammen dürfen daher nicht größer als eine Großseite werden.

Soweit explizit nicht anders angegeben, nimmt der K-Bereich alle Konstanten auf.

3.1.2. Variablenbereich

Im Unterschied zum K-Bereich ist der V-Bereich nicht schreibgeschützt; sonst hat er die gleichen Adressierungsbedingungen.

Der Variablenbereich dient dazu, während des Laufs des Programms Daten aufzunehmen. Dazu kann mit Hilfe des

Pseudobefehls ASP Speicherplatz freigehalten werden. Der Befehl ASP hat die Form

ASP h h: Anzahl der Halbwörter, die freigehalten werden sollen.

Es ist jedoch auch möglich, Teilseiten freizuhalten. Der Befehl hat die Form

ASP p Ka p: Anzahl der Teilseiten
 a: Teil einer Seite
 leer = Ganzseite 2048 Halbwörter
 2 = Halbseite 1024 Halbwörter
 4 = Viertelseite 512 Halbwörter
 8 = Achtelseite 256 Halbwörter

Der Pseudobefehl ASP ist eine Anweisung an den Assembler, die angegebene Anzahl Halbwörter im V-Bereich freizuhalten. Weitere Einzelheiten siehe im Abschnitt Pseudobefehle.

Ist zu erwarten, daß der V- und K-Bereich zusammen mehr als 65 536 Halbwörter benötigen, so ist es zweckmäßig, die Daten, die bereits mit Hilfe von Modifizier- oder Ersetzbefehlen angesprochen werden, in den D-Bereich zu legen.

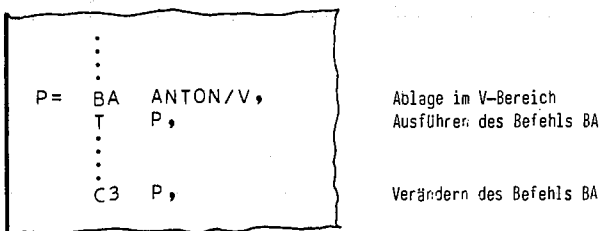


Bild 3.3 Befehl im V-Bereich

Befehle werden im allgemeinen im B-Bereich, also schreibgeschützt, abgelegt. Soll der Befehl veränderlich sein, so kann er mit Hilfe der Spezifikation V in den V-Bereich gelegt werden und mit Hilfe des Befehls T, der an Stelle dieses Befehls steht, zur Ausführung gelangen.

3.1.3. Befehlsbereich

Der Befehlsbereich ist schreibgeschützt. Alle Adresszonen, die der Adressierungsbedingung dieses Bereichs zugehören, müssen innerhalb einer Großseite untergebracht werden, dabei ist es gleich, welche Großseite dies ist. Innerhalb der gleichen Großseite müssen auch die Adresszonen untergebracht werden, die den Adressierungsbedingungen des BV-Bereichs genügen.

Der B-Bereich kann nur mit 22-Bit-Adressen angesprochen werden.

Der B-Bereich nimmt, falls explizit nicht anders angegeben, alle Befehle auf. Für den größten Teil aller Sprungbefehle ist eine Sonderregelung getroffen. Ihre Adressen verstehen sich immer relativ zum Anfang einer Großseite. Solange alle Befehle im B-Bereich (und BV-Bereich) liegen und nicht mehr als 65 536 Befehle vorhanden sind, bedeutet dies keine zusätzliche Belastung für den Benutzer, da im Adressenteil der Sprungbefehle Adressen stehen, die auf Befehle weisen, also sich innerhalb des B-Bereichs und damit in der gleichen Großseite befinden.

Erst wenn die vorgenannte Grenze überschritten wird, müssen die Befehle auf mehrere Großseiten verteilt werden. Dies kann z.B. durch die Definition einer expliziten Zone mit der Adressierungsbedingung des F- oder FV-Bereichs erreicht werden.

Erst wenn Befehle in verschiedenen Großseiten liegen, ist es erforderlich, von einer Großseite in eine andere zu springen. Dazu stehen die Befehle SE, SFBE und SUE zur Verfügung. Des weiteren kann auch mit den anderen Sprungbefehlen in eine andere Großseite gesprungen werden, wenn sie als Zweitbefehl bei den Befehlen MABI oder MU verwendet werden.

Daß Sprungadressen relativ zum Anfang einer Großseite gelten, wird intern dadurch erreicht, daß nicht alle 24 Binärstellen des Befehlsfolgezählers auf die Zieladresse neu gesetzt werden. Es werden vielmehr nur die rechten 16 Bits neu gesetzt, während die linken 8, die die Großseite angeben, unverändert bleiben. Bei den Befehlen SE, SUE, SFB und bei den Sprungbefehlen, die bei den Befehlen MABI und MU als Zweitcode stehen, werden jedoch alle 24 Bits neu gesetzt.

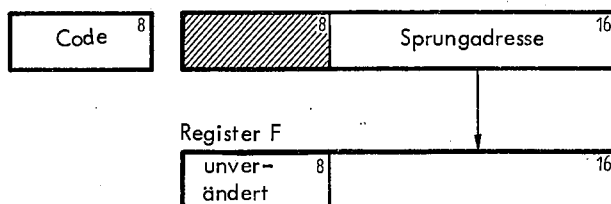


Bild 3.4 Wirkung der meisten Sprungbefehle

3.1.4. Datenbereich

Der D-Bereich ist nicht schreibgeschützt und kann nur über 22-Bit-Adressen angesprochen werden. Er kann in beliebigen Großseiten liegen und auch über Großseitengrenzen hinausgehen.

Mit Hilfe des Pseudobefehls DSP kann im D-Bereich Speicherplatz freigehalten werden. Er hat die Form

DSP h h: Anzahl der Halbwörter, die freigehalten werden sollen.

DSP p Ka p: Anzahl der Teilseiten
 a: Teil einer Seite
 leer = Ganzseite 2048 Halbwörter
 2 = Halbseite 1024 Halbwörter
 4 = Viertelseite 512 Halbwörter
 8 = Achtelseite 256 Halbwörter

Der Pseudobefehl DSP ist eine Anweisung an den Assembler, die angegebene Anzahl Halbwörter im V-Bereich freizuhalten. Weitere Einzelheiten siehe im Abschnitt Pseudobefehle.

Der Datenbereich wird nur dann benötigt, wenn K- und V-Bereich zusammen mehr als 65 536 Halbwörter benötigen. Er ist dann die Verlängerung des V-Bereichs. Daten des D-Bereichs können nur über 22-Bit-Adressen angesprochen werden, d.h. mit Hilfe von Ersetz- oder Modifizierbefehlen.

Ist die Anzahl der benötigten Halbwörter unbekannt, kann sie jedoch an die oben erwähnte Grenze kommen, so ist es zweckmäßig, alle die Daten, die sowieso über Ersetz- und Modifizierbefehle angesprochen werden, in den D-Bereich zu legen.

Auch der Indexbereich kann ohne zusätzliche Arbeit in den D-Bereich gelegt werden.

3.1.5. BV-Bereich

Der BV-Bereich ist nicht schreibgeschützt; sonst gelten die gleichen Bedingungen wie für den B-Bereich. Der BV-Bereich muß zusammen mit dem B-Bereich innerhalb einer Großseite liegen.

Der BV-Bereich kann dazu verwendet werden, um einen Teil der Befehle variabel (also ohne Schreibschutz) zu bekommen.

3.1.6. DK-Bereich

Der DK-Bereich ist schreibgeschützt; sonst hat er die gleichen Bedingungen wie der D-Bereich.

Der DK-Bereich kann dazu verwendet werden, um umfangreiche Konstantenmengen unterzubringen, die die Großseite 0 nicht aufnehmen kann.

3.1.7. F- und FV-Bereich

Der F- und der FV-Bereich entsprechen den B- bzw. BV-Bereichen und haben die gleichen Adressierungsbedingungen.

Während alle Adressenzonen, die zum B- und BV-Bereich gehören, in ein und derselben Großseite abgelegt werden müssen, ist diese Bedingung beim F- und FV-Bereich nicht gegeben. Hier besteht die Forderung, daß jede der Adressenzonen, die im F- und FV-Bereich liegen, für sich innerhalb einer Großseite liegen muß; es kann also jede dieser Zonen in einer anderen Großseite liegen.

Diese Adressierungsbedingungen sind also geeignet, Befehlsfolgen, die eine Großseite überschreiten, auf mehrere Großseiten zu verteilen. Ein Sprung von einer Adressenzone des F- oder FV-Bereichs in eine andere Adressenzone dieses Bereichs oder in eine der Adressenzonen des B- oder BV-Bereichs und umgekehrt, muß stets ein Sprungbefehl sein, der in eine andere Großseite springen kann. Bei Sprüngen zwischen den Adressenzonen des B- und BV-Bereichs ist dies nicht nötig.

3.1.8. I-Bereich

Der I-Bereich ist nicht schreibgeschützt. Er ist mit 16-Bit-Adressen adressierbar.

Der Anfang des Indexbereichs wird mindestens 256 Adressenstellen vor dem Ende des Gebiets abgelegt, in dem der Indexbereich angeordnet wird.

3.2. Adressenzonen

Die Informationseinheiten eines Quellenprogramms werden vom Assembler sortiert und entsprechend ihren Adressierungsbedingungen bestimmten Adressenzonen zugeordnet.

Implizit gibt es 13 Adressenzonen. Sie werden nur bei Bedarf vom Assembler eingerichtet. Des weiteren können explizit durch die Pseudobefehle ZONE, FZONE und CZONE weitere Adressenzonen eingerichtet werden.

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

Jede Adressenzone beginnt mit der relativen Adresse 0. Bei der Montage werden die Adressenzonen durch Addieren einer Translationsgröße in den Adressenraum des Operators gemäß den Adressierungsbedingungen der Zone abgebildet. Nach der Montage werden die Adressenzonen vergessen.

Die Anzahl der explizit und implizit definierten Adressenzonen ist pro Montageobjekt auf 31 begrenzt.

3.2.2. Explizite Adressenzonen

Neben den impliziten Adressenzonen können bei Bedarf explizit weitere Adressenzonen eingerichtet werden. Dazu dienen die Pseudobefehle

name = ZONE a	name: Name, den die Zone erhalten soll
name = CZONE a	a: Adressierungsbedingung für die
name = FZONE a	K, B, DK, F, FV, V, BV, D, FV oder I

Die eingerichtete Zone bekommt den angegebenen Namen. Die Adressierungsbedingung der Zone entspricht dem in Abschnitt 3.1. Beschriebenen.

```

ANH=  ZONE  FV,
XB3=  CZONE  V,
B13=  FZONE  DK,

```

Bild 3.5 Beispiele für explizite Adressenzonen

3.2.1. Implizite Adressenzonen

Es gibt 13 implizite Adressenzonen. Sie werden bei Bedarf vom Assembler eingerichtet. Sie erhalten keine Namen und werden für Beschreibungszwecke und für die Verwendung bei Pseudobefehlen durch zwei Zeichen dargestellt.

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

Jeweils das erste Zeichen gibt die Adressierungsbedingung der Zone an, wie sie im Abschnitt 3.1. beschrieben ist. Es wird Ablageschlüssel genannt.

Das zweite Zeichen ist eine Kennzeichnung in bezug auf Literale. Es wird Literalschlüssel genannt und hat folgende Bedeutung:

- 0 Informationseinheiten außerhalb von Literalen
- 1 Informationseinheiten in Literalen (Literalschlüssel 1. Ordnung)
- 2 Informationseinheiten in Literalen, die wiederum in Literalen stehen (Literalschlüssel 2. Ordnung)
- G Ganzwortliterals; Literale, die ein Ganzwort einnehmen und im K-Bereich liegen, werden bei mehrfachem Auftreten nur einmal abgelegt (nur im K-Bereich möglich).

Die Informationseinheiten werden vom Assembler, sofern nicht explizit anders bestimmt, einer dieser Adressenzonen zugeordnet. Die vorstehend genannten Kennzeichnungen der Adressenzonen sind daher auch gleichzeitig die implizite Ablagekennung einer Informationseinheit.

Diese implizit einer Informationseinheit zugehörige Ablagekennung wird also aus dem Ablageschlüssel und dem Literalschlüssel gebildet. Der Ablageschlüssel kann explizit mit Hilfe des Pseudobefehls STARR oder durch eine Ablagespezifikation geändert werden (siehe auch Abschnitt 3.3.2. und 3.3.3.).

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

Daraus resultiert, daß einer Commonzone nur in einem der zu einem Operator gehörenden Quellenprogramme Informationseinheiten zugeordnet werden dürfen. Darüber hinaus stehen nur noch die Pseudobefehle ASP und DSP zur Verfügung, um die Länge der Commonzone für das jeweilige Quellenprogramm festzulegen.

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

Der mit dem FZONE-Befehl freigehaltene Speicherraum steht erst dann zur Verfügung, wenn dies dynamisch gefordert wird. Sinnvoll ist die Verwendung einer Freihaltezone dann, wenn in Verbindung mit dem Pseudobefehl LUECKE gearbeitet und über den Pseudobefehl GEBAN eine Verbindung mit einem Gebiet hergestellt wird (siehe Abschnitt 3.4.3.).

Die Definition von expliziten Adressenzonen ist nur in Verbindung mit dem Pseudobefehl ABLAGE sinnvoll. Beispiele zu den Befehlen ZONE, CZONE und FZONE sind im Abschnitt 3.3.4. gegeben.

3.3. Ablage der Informationseinheiten

Alle Befehle und Konstanten sowie die Pseudobefehle ASP und DSP haben implizit eine bestimmte Ablagekennung. Gemäß dieser impliziten Ablagekennung werden diese Informationseinheiten bestimmten Adressenzonen zugeordnet.

Den vorstehend genannten Informationseinheiten kann einer der Buchstaben K, V, B oder D als Ablagespezifikation nachgestellt werden. Sie bewirkt, daß der Ablageschlüssel der Ablagekennung gemäß der Spezifikation geändert wird.

Soll für mehrere hintereinanderliegende Informationseinheiten der Ablageschlüssel geändert werden, so ist dies mit dem Pseudobefehl STARR möglich.

Die beiden vorstehenden Möglichkeiten verändern die Ablagekennung von Informationseinheiten und damit auch ihre Ablage in einer Adressenzone. Sollen Informationseinheiten nicht in der ihnen zugehörigen impliziten Adressenzone, sondern in einer explizit vereinbarten Adressenzone abgelegt werden, so ist dies mit dem Pseudobefehl ABLAGE möglich.

Bei der Assemblierung wird eine Informationseinheit gemäß ihrer Ablagekennung der gleichnamigen Adressenzone zugeordnet. Hat also z.B. eine Informationseinheit die Ablagekennung V0, so wird sie auch der Adressenzone V0 zugeordnet.

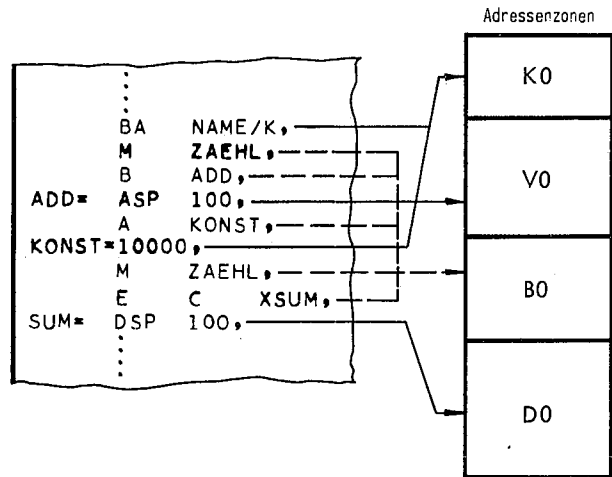


Bild 3.7 Zuordnung zu Adressenzonen

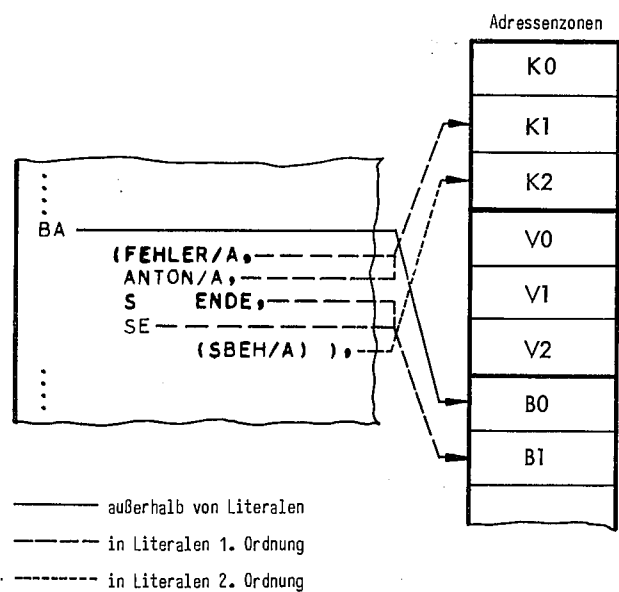
3.3.1. Implizite Ablagekennung

Befehle, Konstanten und die Pseudobefehle ASP und DSP besitzen implizit die in Bild 3.6 gezeigte Ablagekennung.

	außerhalb von Literal	in Literal 1. Ordnung	in Literal 2. Ordnung	Ganzwörter
Konstanten	K0	K1	K2	KG
Befehle	B0	B1	B2	
Pseudobefehl ASP	V0	V1	V2	
Pseudobefehl DSP	D0	D1	D2	

K, V, B und D: Ablageschlüssel
0, 1, 2 und G: Literalschlüssel

Bild 3.6. Implizite Ablagekennung



— außerhalb von Literalen
- - - in Literalen 1. Ordnung
- - - - in Literalen 2. Ordnung

Bild 3.8. Zuordnung bei Literalen

3.3.2. Ablagespezifikation

Soll eine Informationseinheit anders abgelegt werden als implizit vorgegeben, so kann ihr eine Ablagespezifikation angefügt werden (siehe Abschnitt 5.5.). Als Ablagespezifikation können die Buchstaben K, V, B oder D angegeben werden. Sie entsprechen den Adressierungsbedingungen in Abschnitt 3.1.

Durch diese Ablagespezifikation wird bei der implizit vorgegebenen Ablagekennung der Ablageschlüssel (erstes Zeichen) gemäß der Spezifikation geändert. So wird z.B. die implizite Ablagekennung K0 der Konstanten

0.361/V,

durch die Ablagespezifikation V in die Ablagekennung V0 geändert. Damit ist diese Konstante im V-Bereich und nicht mehr schreibgeschützt.

Der Literalschlüssel bleibt durch die vorstehend erläuterte Ablagespezifikation unverändert. Lediglich der Literalschlüssel G, der im B-, V- und D-Bereich nicht vorkommt, wird sinngemäß auf 0, 1 oder 2 geändert.

		Ablagekennung
	'',00 DM''/V,	K0 → V0
T	BEF,	B0
BEF= BA	'DM''/V,	B0 → V0
B		B0
	(''000000''/V),	KG → V1

Bild 3.9 Änderung der Ablagekennung

kennung zu ändern. Dabei gibt die mit s angegebene Ablagespezifikation an, wie der Ablageschlüssel sein soll. Der STARR-Befehl wirkt auf alle folgenden Informationseinheiten so, als ob jede die mit s angegebene Spezifikation erhalten hätte (siehe Abschnitt 3.3.2.).

Ein besonderer Fall ist zu beachten. Folgt auf einen benannten Pseudobefehl SEGMENT unmittelbar ein Pseudobefehl STARR, so wird im allgemeinen der Name des Segments nicht einer Informationseinheit derselben Zone zugeordnet, der die folgenden Informationseinheiten zugeordnet werden. Dadurch kann man die folgenden Informationseinheiten nicht mit dem Namen des Pseudobefehls SEGMENT aufrufen. Dies kann dadurch umgangen werden, daß man den Pseudobefehl STARR vor den benannten Pseudobefehl SEGMENT legt.

Beendet wird die Wirkung des STARR-Befehls durch einen neuen STARR-Befehl oder durch den Pseudobefehl

STEND

Er ist eine Anweisung an den Assembler, die durch einen vorhergehenden STARR-Befehl gegebene Anweisung aufzuheben.

Wird innerhalb des Bereichs, in dem ein STARR-Befehl gilt, eine Informationseinheit mit der Ablagespezifikation gemäß Abschnitt 3.3.2. versehen, so gilt diese. Die explizite Angabe einer Ablageanweisung bei der Informationseinheit überschreibt also die Angabe durch den Pseudobefehl STARR.

Damit können Befehle, Konstanten und die durch die Pseudobefehle ASP und DSP freigehaltenen Speicherbereiche auf drei Arten ihren Ablageschlüssel erhalten.

...			
ZU	255,	B0	
ADR=	DBER/AV,	V0	
	STARR D,	-	} STARR D
DBER=	ANTON/A,	D0	
	TCB ANF,	D0	
ANF=	ASP 200/V,	V0	
	XBA EMIL,	D0	
	STEND,	-	
	SA ADR,	B0	
	SFB UPR,	B0	
...			

Bild 3.10. Beispiel für den STARR-Befehl

3.3.3. Pseudobefehl STARR

Soll eine Reihe von aufeinanderfolgenden Informationseinheiten eine andere Ablagekennung erhalten, so braucht nicht jede Informationseinheit mit einer Ablagespezifikation versehen zu werden, sondern es kann der Pseudobefehl STARR verwendet werden.

STARR s s: Ablagespezifikation
 K = Konstantenbereich
 V = Variablenbereich
 B = Befehlsbereich
 D = Datenbereich

Der STARR-Befehl ist eine Anweisung an den Assembler, bei allen folgenden Informationseinheiten die Ablage-

- Der implizite Ablageschlüssel
 Befehle in den B-Bereich
 Konstanten in den K-Bereich
 Speicherplatz durch Pseudobefehl ASP im V-Bereich
 Speicherplatz durch Pseudobefehl DSP im D-Bereich
- Der Ablageschlüssel durch Pseudobefehl STARR.
 Er überschreibt den impliziten Ablageschlüssel.
- Der explizite Ablageschlüssel durch Angabe einer Ablagespezifikation bei einer Informationseinheit (Befehl, Konstante, Pseudobefehl ASP und DSP). Er überschreibt die implizite Zuordnung und die durch den Pseudobefehl STARR gegebene Zuordnung.

Durch diese drei Möglichkeiten erhält ein Befehl oder eine Konstante sowie der durch die Pseudobefehle ASP und DSP freigehaltene Speicherplatz einen Abtageschlüssel. Der Literalschlüssel wird stets implizit vorgegeben. Mit Abtageschlüssel und Literalschlüssel hat die Informationseinheit ihre Abtagekennung.

3.3.4. Pseudobefehl ABLAGE

Ohne explizite Angabe wird eine Informationseinheit gemäß ihrer Abtagekennung, die sie implizit oder explizit erhalten hat, der entsprechenden impliziten Adresszone zugeordnet. Dabei ist jeder Abtagekennung eine implizite Adresszone zugeordnet.

Sollen nun Informationseinheiten nicht der zugehörigen impliziten Adresszone, sondern einer mit den Pseudobefehlen ZONE, CZONE oder FZONE vereinbarten expliziten Adresszone zugeordnet werden, so ist dies mit dem Pseudobefehl

```

ABLAGE zone (k)   zone: Name einer explizit vereinbarten
                   Adresszone
                   k: Abtagekennung
                   KO VO BO DO
                   K1 V1 B1 D1
                   K2 V2 B2 D2

```

mehrere Angaben durch Komma trennen

möglich. Alle Informationseinheiten, die ohne den Befehl ABLAGE den mit k angegebenen Adresszonen zugeordnet worden wären, werden jetzt der expliziten Adresszone mit dem Namen zone zugeordnet. Für k können mehrere Abtagekennungen angegeben werden, die durch Komma zu trennen sind. Die explizite Zone muß vorher vereinbart sein.

Ein ABLAGE-Befehl legt für die unter k angegebene Abtagekennung eine neue Zuordnung fest, hebt also eine vorhergehende (auch von einem ABLAGE-Befehl stammende) Zuordnung auf.

Mit dem Pseudobefehl AEND wird die Zuordnung der Informationseinheiten mit bestimmter Abtagekennung wieder auf die impliziten Adresszonen zurückgeführt. Er hat die Form

```

AEND (k)         k: Abtagekennung
                  KO VO BO DO
                  K1 V1 B1 D1
                  K2 V2 B2 D2

```

mehrere Angaben durch Komma trennen

Der Befehl beendet damit für die angegebenen Abtagekennungen die durch den ABLAGE-Befehl angegebene Zuordnung zu explizit definierten Adresszonen.

Es ist zu beachten, daß bei der Definition der expliziten Adresszone eine Adressierungsbedingung angegeben wurde. Alle Informationseinheiten, die einer Adresszone durch den Pseudobefehl ABLAGE zugeordnet sind, werden gemäß der Adressierungsbedingung der Zone behandelt, auch wenn sie implizit anders ist. Wird eine Konstante, die implizit die Abtagekennung KO hat, in einer Adresszone angeordnet, die die Adressierungsbedingung D hat, so kann sie nicht mit 16-Bit-Adressen adressiert werden. Ein Fehler dieser Art wird durch den Assembler nicht gemeldet, da er nicht feststellen kann, ob eine Modifizierung vorgenommen wurde.

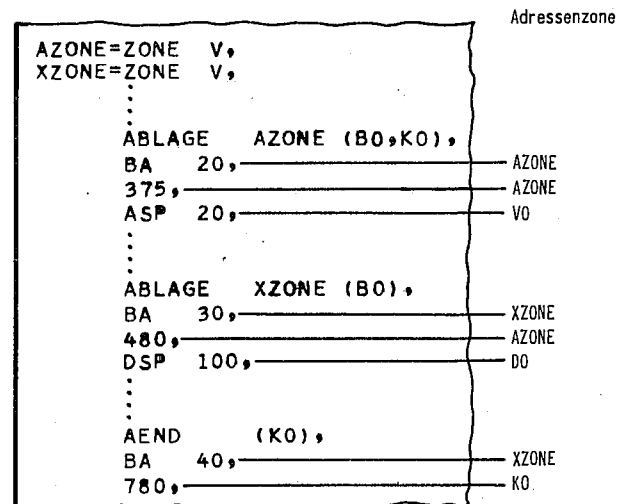
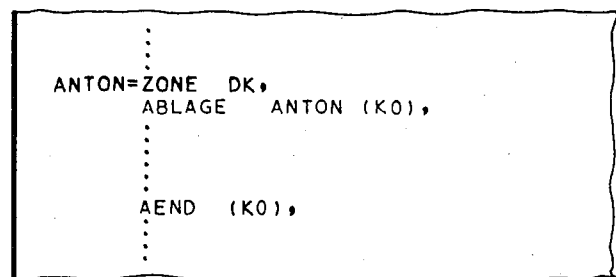


Bild 3.11 Beispiel für die Zuordnung zu explizit definierten Adresszonen



Die zwischen "ABLAGE" und "AEND" liegenden Konstanten wurden in der expliziten Zone ANTON abgelegt, die durch 22 Bit-Adressen zu adressieren ist.

Bild 3.12. Beispiel für die Anordnung von Konstanten schreibgeschützt im D-Bereich

Quellenprogramm 1

```
ALLG= CZONE V,  
      ABLAGE ALLG (V0),  
      DSP 500/V,  
      AEND (V0),  
      .  
      .  
      .
```

Quellenprogramm 2

```
ALLG= CZONE V,  
      ABLAGE ALLG (V0),  
      ASP 1K,  
      AEND (V0),  
      .  
      .  
      .
```

Die Commonzone ALLG wird 1024 Garzwörter lang.

Bild 3.13 Beispiel für eine Commonzone

```
FREI= FZONE D,  
      ABLAGE FREI (D0),  
      DSP 5K4,  
      AEND (D0),  
      .  
      .  
      .
```

Bild 3.14. Beispiel für eine Freihaltezone

3.3.5. Reihenfolge der Ablage

Für die Ablage der Informationseinheiten kann es interessant sein, in welcher Reihenfolge sie abgelegt werden.

Der Assembler verarbeitet die Informationseinheiten in der statischen Reihenfolge, in der sie in den Rechner eingegeben werden. In der gleichen Reihenfolge werden sie auch in den Adresszonen abgelegt.

Bei Literalen wird diese Reihenfolge ebenfalls eingehalten. Es wird also erst der Befehl verarbeitet, in dem das Literal vorkommt und danach die Informationseinheiten dieses Literals.

Bei Makro-Aufrufen wird so verfahren, als ob das aufgerufene Programmstück (Makro) an Stelle des Makroaufrufs steht.

Wird Ersetzungstechnik verwendet, so wird sie in einem Vorlauf des Assemblers ausgeführt, und erst danach beginnt die Assemblierung.

Werden mit dem ABLAGE-Befehl Informationseinheiten verschiedener Ablagekennung einer Adresszone zugeordnet, so werden alle Informationseinheiten, die dieser Zone zugeordnet sind, in der Reihenfolge der Verarbeitung angeordnet; sie sind also nicht nach Ablagekennung geordnet.

3.4. Adresszonen im Adressraum

Bei der Montage der übersetzten Quellenprogramme zu einem Operator, werden die Adresszonen der einzelnen Quellenprogramme Gebieten zugeordnet, und diese Gebiete werden im Adressraum des Operators angeordnet (siehe Bild 3.1).

Das Montageprogramm entscheidet selbsttätig auf Grund der Adressierungsbedingungen und gewisser Optimierungsbedingungen, wieviel Gebiete angelegt, wie die Adresszonen in den Gebieten und wie die Gebiete im Adressraum des Operators angeordnet werden. Der Benutzer braucht hierzu keine Angaben zu machen.

Wünscht der Benutzer jedoch eine bestimmte Anordnung im Adressraum, so kann er dies explizit mit Hilfe von Pseudobefehlen angeben. Zu diesem Zweck ist es erforderlich, mit dem Pseudobefehl GEBIET ein Gebiet zu eröffnen. Dem Gebiet können mit Hilfe des Pseudobefehls ZONAN Adresszonen zugeordnet werden.

Freihaltezonen, die durch den Pseudobefehl FZONE vereinbart wurden, können nicht einem Gebiet zugeordnet werden. Für diese Zonen wurde der Begriff Lücke geschaffen. Mit Hilfe des Pseudobefehls LUECKE wird eine "Lücke" geschaffen, der mit dem Pseudobefehl ZONAN die Freihaltezonen - und nur diese - zugeordnet werden können.

Für alle explizit vereinbarten Gebiete und Lücken muß explizit mit dem Pseudobefehl GEBAN die Anordnung der Gebiete und Lücken im Adressraum festgelegt werden.

Für alle Adresszonen, die nicht explizit einem Gebiet (oder einer Lücke) zugeordnet und damit auch im Adressraum angeordnet sind, übernimmt das Montageprogramm die Zuordnung.



3.4.1. Pseudobefehl GEBIET

Durch den Pseudobefehl GEBIET wird ein Gebiet explizit vereinbart.

GEBIET (p) p: Gebietsparameter,
mehrere Angaben
durch Komma trennen

Es können mehrere Gebietsparameter angegeben werden. Die Angabe der Gebietsparameter ist nicht zwingend. Mehrere Parameter können in beliebiger Reihenfolge geschrieben werden, wenn sie mit ihrem Namen angegeben werden. Wird aber der Name mit dem Gleichheitszeichen fortgelassen, dann wird die nachfolgende Reihenfolge der Parameter vorausgesetzt. Es ist dadurch eine mehrfache Besetzung eines Parameters möglich. In einem solchen Fall ist der letzte Parameterwert gültig. Es wird dann eine Warnung ausgegeben. Folgende Gebietsparameter sind zulässig:

p₀ : operatorspezifischer Gebietsname
ONAME = name name: operatorspezifischer
Gebietsname

Der operatorspezifische Gebietsname muß eindeutig sein in allen zum Operator gehörenden Montageobjekten. Es wirken nur die ersten 6 Zeichen unterscheidend.

p₁ : prozeßspezifischer Gebietsname
PNAME = name name: prozeßspezifischer
Gebietsname

Zusätzlich zum operatorspezifischen Gebietsnamen oder stattdessen kann auch ein prozeßspezifischer Gebietsname angegeben werden. Dieser muß in dem Prozeß, zu dem der Operator gehört, eindeutig sein. Mindestens einer der Parameter p₀ und p₁ muß angegeben werden. Es wirken nur die ersten 6 Zeichen unterscheidend.

p₂ : Lebensdauer des Gebietes
LD = d d: Lebensdauer des Gebietes
L = Laufzeitgebiet
D = Dauergebiet
Parameter ist mit D vorbesetzt

p₃ : Adressierungsbedingung
ADR = a a: Adressierungsbedingung
K, B, DK, F
V, BV, D oder FV
Parameter ist mit K vorbesetzt

Alle Gebiete mit ADR = B oder ADR = BV werden derselben Großseite zugeordnet wie die B-Adressenzonen.

p₄ : gewünschte Verarbeitungsklasse
VK1 = v
p₅ : geforderte Verarbeitungsklasse
VK2 = v v: Verarbeitungsklasse
KS = Kernspeicher
MK = Massenkernspeicher, nicht direkt adressierbar
TR = Trommel
PL = Platte
Beide Parameter sind mit KS vorbesetzt

p₆ : Lagerklasse
LK = k k: Lagerklasse
MK = Massenkernspeicher,
nicht direkt adressierbar
TR = Trommel
PL = Platte
HG = Hintergrundspeicher
Parameter ist mit PL vorbesetzt

p₇ : Vorbesetzung
VB = (t, 's') t: Typenkennung
0, 1, 2, 3
s: 12 Sedezimalzeichen

Mit diesem Parameter kann die Vorbesetzung jedes Ganzwortes im Gebiet angegeben werden. Diese Vorbesetzung überschreibt für dieses Gebiet die allgemeine Vorbesetzung, die mit dem Pseudobefehl VORBES eingestellt wird.

p₈ : Paßwort für Lesezugriff
PWL = b b: Bedingung
name = Paßwort für Lesezugriff
0 = nicht als Fremdgebiet
beschaffbar
leer = als Fremdgebiet ohne
Paßwort beschaffbar

```
GEBIET ( ANTON ),
GEBIET ( ONAME=AGW, PNAME=AGWERT, LD=L, ADR=D, VK1=KS, VK2=TR, LK=PL,
        VB=( 3, '000000000000' ), PWL=LES, PWLS=SCHREIB ),
GEBIET ( AGW, AGWERT, L, D, KS, TR, PL, ( 3, '000000000000' ), LES, SCHREIB ),
GEBIET ( AGW, AGWERT, L, D, VK2=TR, PL, ( 3, '000000000000' ), LES, SCHREIB ),
GEBIET ( VB=( 3, '000000000000' ), ONAME=AGW, AGWERT, L, D, KS, TR, PL, LES,
        SCHREIB ),
```

Die Gebiete AGW sind identisch

Bild 3.15 Beispiele für Pseudobefehl GEBIET

Einem Gebiet darf eine Freihaltezone nicht zugeordnet werden; sie kann nur einer Lücke zugeordnet werden. Umgekehrt dürfen einer Lücke nur Freihaltezonen zugeordnet werden. Für diese Zuordnung wird sinngemäß der gleiche Befehl verwendet

ZONAN lücke (fzone) lücke: Name der Lücke
 fzone: Namen der Freihaltezonen,
 die dem Gebiet zugeordnet
 werden sollen (mehrere
 durch Komma trennen).
 Nur Freihaltezone erlaubt.

Siehe auch Abschnitt 3.4.2.

Alle in einem Pseudobefehl ZONAN auftretenden Zonen und das Gebiet bzw. die Lücke müssen vorher im Quellenprogramm deklariert sein (durch die Pseudobefehle ZONE, GEBIET bzw. LUECKE).

```

:
:
ZONAN GEB1 (ZONE1,ZA,ZB),
:
ZONAN GEB1 (ZON8,ZX,B13),
:
ZONAN FREI (F1,FREI5),
:

```

Bild 3.17. Beispiel für ZONAN-Befehle

3.4.4. Pseudobefehl GEBAN

Der Pseudobefehl GEBAN ist eine Anweisung an das Montageprogramm, die explizit mit dem Pseudobefehl GEBIET definierten Gebiete und die explizit mit dem Pseudobefehl LUECKE vereinbarten Lücken im Adressenraum anzuordnen. Für alle expliziten Gebiete und Lücken ist diese Zuordnung zwingend.

GEBAN bezug (liste)

bezug: Anfang der Gebietszuordnung im
 Adressenraum;
 Seitennummer (dezimal oder sedezial)
 Gebietsname oder
 Lückename
 liste: Gebietsname,
 Lückename
 Dezimalzahl oder
 Sedezialzahl
 (mehrere Angaben durch Komma trennen)

Der Pseudobefehl bewirkt, daß die innerhalb der Klammern aufgelisteten Gebiets- und Lückennamen in dieser Reihenfolge im Adressenraum angeordnet werden. Der Anfangspunkt für die Anordnung wird durch den mit bezug angegebenen Bezugsparameter angegeben. Er kann ein absoluter Bezug oder ein Namensbezug sein.

Der absolute Bezug kann als Dezimalzahl oder Sedezialzahl angegeben werden. Eine Sedezialzahl ist in Apostrophe einzuschließen. In beiden Fällen ist die Zahl die Seitennummer und gibt an, mit welcher Seite die Zuordnung beginnt. Die Seitennummer ist die Adresse des ersten Halbworts einer Seite dividiert durch 2048 (2^{11}).

Der Namensbezug kann ein Gebiets- oder Lückename sein. Er muß in einem vorhergehenden Pseudobefehl GEBAN bereits zugeordnet sein, damit sich aus dem Namensbezug ein absoluter Bezug ableiten läßt. Der Namensbezug gibt an, daß die Zuordnung mit der ersten Seite hinter dem Gebiet fortgesetzt wird, das diesen Namensbezug nennt. Es können auch Namen angegeben werden, die in einem anderen Quellenprogramm definiert wurden, wenn das Quellenprogramm mit zum Operator gehört.

Die Namenliste kann sowohl Gebiets- als auch Lückennamen in beliebiger Reihenfolge enthalten. Sie sind durch Komma zu trennen. Darüber hinaus kann auch an Stelle eines Lückennamens eine Freihaltezahl als Dezimal- oder Sedezialzahl angegeben werden. Sie gibt an, wieviel Seiten im Adressenraum freigehalten werden sollen. Ein physikalischer Speicherraum wird dabei noch nicht beansprucht. Hiermit wird eine Lücke definiert, die keinen Namen bekommt.

```

:
:
GEBAN 0 (G1,G2,G3),
:
:
GEBAN G2 (G7),
GEBAN G3 (G4,2,G5),
:
:
GEBAN '20' (G8,5,G9,G10),
:
:
GEBAN 16 (G11,G12),
GEBAN G4 (G6),
:
:
GEBAN G11 (G13),
GEBAN G11 (G14,G15),
:
:

```

Bild 3.18. Beispiel zum Befehl GEBAN

Alle explizit definierten Gebiete und Lücken müssen einmal und nur einmal im Adressenraum eines Operators angeordnet werden. Die einzelnen Gebiete und Lücken dürfen sich auch im Adressenraum überschneiden, jedoch darf dann nur eines der Gebiete die geforderte Verarbeitungsklasse Kernspeicher haben (VK2 = KSP).

Bezüglich der Adressierungsbedingungen gilt folgendes: Steht die Adressierungsbedingung eines Gebietes im Widerspruch zu seiner Anordnung im Adressenraum, so wird eine Fehlermeldung gegeben und die Montage abgebrochen (z.B. Gebiet mit ADR = K mit Adresse größer 2^{16}).

Mit der Anordnung des Gebietes im Adressenraum werden auch die Adresszonen, die dem Gebiet zugeordnet sind, im Adressenraum angeordnet. Wird dabei die Adressierungsbedingung der Adresszone verletzt, so wird eine Fehlermeldung gegeben und die Montage abgebrochen. Es sei darauf hingewiesen, daß die Adressierungsbedingung der Zone nicht mit der des Gebietes übereinzustimmen braucht. Es kann z.B. die Adresszone die Adressierungsbedingung F haben und für das Gebiet ADR = DK sein. Die Bedingungen sind dann schon erfüllt, wenn die Adresszonen nicht über eine Großseitengrenze gehen.

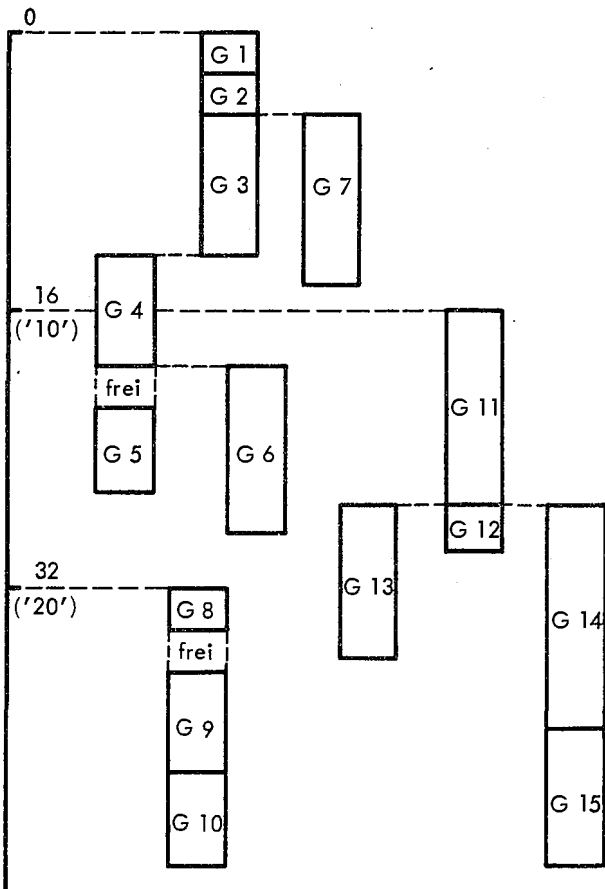


Bild 3.19.. Beispiel einer Anordnung von Gebieten im Adressenraum zu Bild 3.18

3.5. Großseite - Seite

Die Zuteilung des physikalischen Speicherraums an den jeweils rechenwilligen Operator geschieht in Einheiten von 2048 Halb- bzw. 1024 Ganzwörtern. Diese Einheit wird als Seite bezeichnet.

Des weiteren wirkt ein Schreibschutz stets auf eine ganze Seite.

Die Adresse des ersten (Halb-) Wortes einer Seite hat intern die rechten 11 Bits mit Nullen besetzt. Wird z.B. mit dem Pseudobefehl

EMIL = ASP 2K,

ein Speicherbereich reserviert, so hat intern die Adresse EMIL die rechten 11 Bits auf Null gesetzt. Der Anfang einer Seite im Programm fällt also mit dem (physikalischen) Anfang einer Seite im Kernspeicher zusammen.

Bei einer Achtelseite sind entsprechend die rechten 8 Bits der internen Anfangsadresse auf Null gesetzt.

Dies ist für den Transport von Daten innerhalb des Systems von Bedeutung, da Daten innerhalb des Systems in Form von Gebieten transportiert werden. Ein Gebiet ist stets ein ganzzahliges Vielfaches einer Seite. Transporte zwischen den Gebieten können auch in Form von Achtelseiten erfolgen. Hier fällt die Achtelseite des Programms mit der Achtelseite im Kernspeicher zusammen. Es können also nur Achtelseiten transportiert werden, deren Anfangsadresse intern in den rechten 8 Bits mit Null besetzt ist. Die mit den Pseudobefehlen ASP und DSP freigehaltenen Teilseiten erfüllen diese Bedingung. Es ist z.B. durch den Pseudobefehl

ASP 0K8/K

möglich, die folgenden Konstanten mit einer Achtelseitenadresse beginnend, abzulegen.

Der Adressenteil eines Befehls ist 16 Bits lang. Dadurch lassen sich im Adressenteil des Befehls 65 536 Halbwörter, bzw. 32 768 Ganzwörter adressieren. Diese Einheit wird als Großseite bezeichnet. Sie umfaßt 32 Seiten. Diese erste direkt adressierbare Großseite ist die Großseite Nummer 0.

Beim Holen eines Befehls in das Befehlswerk wird der Adressenteil auf 24 Bits erweitert, indem links Null Bits hinzugefügt werden. Damit bleibt die Adresse unverändert. Im Befehlswerk wird dann nur noch mit 24-Bit-Größen gearbeitet, von denen 22 Bits zur Adressierung verwendet werden können. Durch Modifizierung kann die Adresse nun verändert werden, so daß alle 22 Bits verwendet werden können. Damit haben wir 22-Bit-Adressen und können damit auch die Großseiten 1 bis 63 adressieren. Über Ersetzbefehle kann das Vorstehende ebenfalls erreicht werden. Näheres dazu ist dem Abschnitt "Adressrechnung" zu entnehmen.

Wir wollen hier also festhalten, daß mit 16-Bit-Adressen nur die Großseite 0 und mit 22-Bit-Adressen alle Großseiten (auch die Großseite 0) erreicht werden können.

Durch die Tatsache, daß die meisten Sprungbefehle relativ zur Großseite springen, in der sie stehen (siehe Abschnitt 3.3.) und dadurch, daß alle Befehle (im Normalfall) im B-Bereich liegen, wird erreicht, daß die Befehle nicht in der Großseite 0 stehen müssen und trotzdem bei den Sprungbefehlen mit 16-Bit-Adressen gearbeitet werden kann.

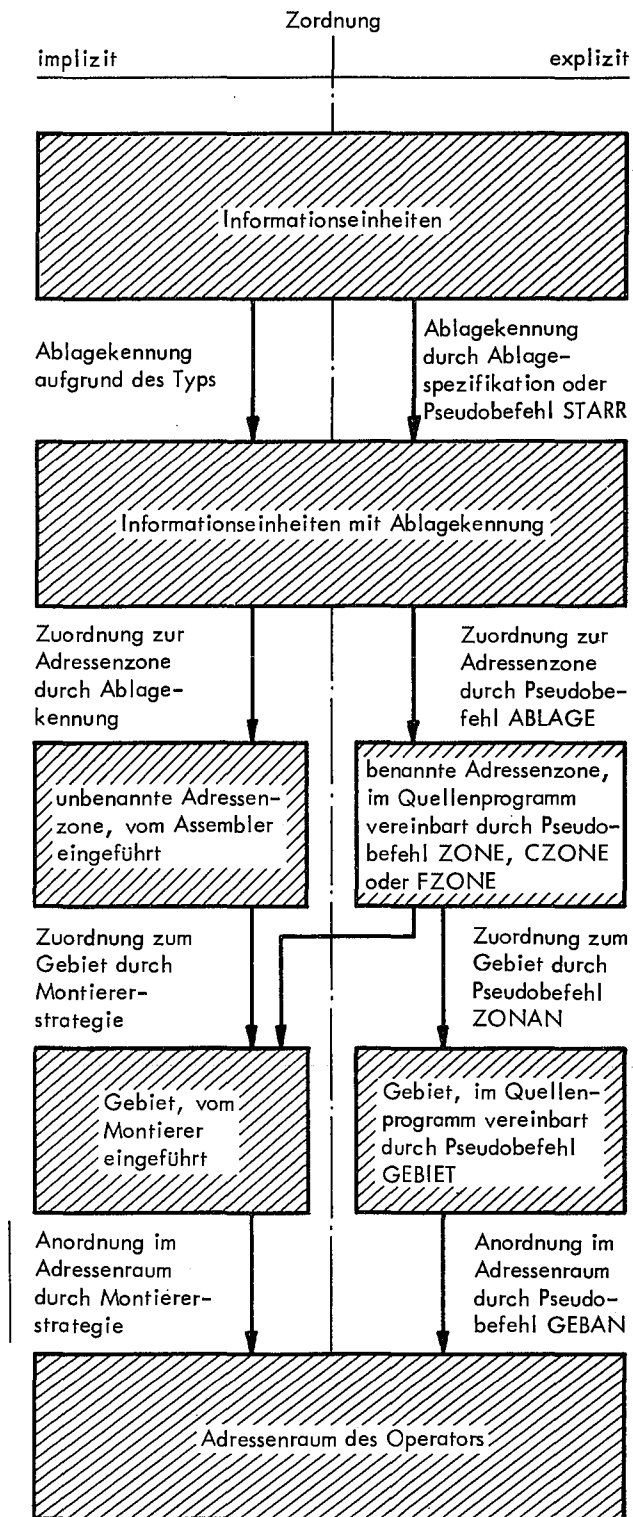


Bild 3.20 Übersicht über die Zuordnung von Informationseinheiten, Adresszonen und Gebieten

4. NIEDERSCHRIFT

B

Die Niederschrift des Quellenprogramms ist an keinerlei Format gebunden. Zwischenräume und Wagenrückläufe können beliebig eingeschoben werden und können dazu dienen, die Niederschrift des Quellenprogramms übersichtlich zu gliedern.

Der Wechsel auf die nächste Zeile wird vom Assembler ignoriert und ist ohne Einfluß auf die Übersetzung. Bei Lochkarten entspricht dies dem Wechsel auf eine neue Karte, beim Fernschreiber dem Wagenrücklauf mit Zeilenvorschub. Auch in Oktadenfolgen, Kommentaren und Überschriften ist der Zeilenvorschub ohne Bedeutung.

Der Zwischenraum kann beliebig oft eingeschoben werden. Er ist zur Trennung der einzelnen Teile bei Befehlen und Pseudobefehlen erforderlich. Im Schriftbild bleibt ein Zwischenraum, d.h. für den Zwischenraum gibt es hier kein spezielles Druckzeichen. Im Einzelfall kann es zweckmäßig sein, dieses Zeichen bildlich darzustellen. Im DIN-Blatt 66 003 wird das Zeichen \bar{b} (von englisch "blank") oder das Zeichen $_$ empfohlen.

4.1. Voreinstellung für Operatoren

Bei der Niederschrift werden stets Quellenprogramme geschrieben. Die Programmiersprache TAS bietet die Möglichkeit, ein Problem in mehrere Teile zu zergliedern und für jeden dieser Teile ein Quellenprogramm zu schreiben, das für sich übersetzt und getestet werden kann. Die Übersetzung erfolgt stets in den Montagecode. Durch ein spezielles Programm, das Montageprogramm, werden die einzelnen Montageobjekte (das sind die in den Montagecode übersetzten Quellenprogramme) zu einem startfähigen Operator montiert.

Auch für den Fall, daß nur ein Quellenprogramm zum Operatorlauf gehört, muß das Montageprogramm das (eine) Montageobjekt zum Operator montieren (!). Das heißt, es muß der Montagecode in den Maschinencode übersetzt werden.

Vor dem Start eines Operators werden bestimmte genau definierte Anfangsbedingungen hergestellt. Die Anfangsbedingungen müssen durch die folgenden sechs Anweisungen angegeben werden. Es sind dies

- die Adresse, auf die das Indexbasisregister X gesetzt werden soll
- die Indexadresse, auf die das Unterprogrammregister U gesetzt werden soll
- die Adresse, auf die beim Auftreten eines Fehlers gesprungen werden soll

- die Adresse, bei der der Operator gestartet werden soll
- der Wert, mit dem reservierte Speicherplätze vorbelegt werden
- die Adresse, bei der der Kontrollblock für Testversionen beginnt.

Diese Angaben werden durch Pseudobefehle gemacht. Das Indexbasisregister wird mit dem Pseudobefehl

`XBASIS m` m: Adresse des Halbwortes,
bei dem der Indexbereich
beginnt.

gesetzt.

Der Indexbereich kann an beliebiger Stelle in einem nicht schreibgeschützten Ablagebereich, also im V- oder D-Bereich, beginnen. Wo er beginnt, gibt die Adresse im Indexbasisregister an. Für den Indexbereich sind soviele Halbwörter freizuhalten, wie Indexzellen verwendet werden. Dies kann mit dem Pseudobefehl ASP oder DSP erfolgen. Ist die Anzahl der Indexzellen nicht bekannt, so kann eine genügend große Anzahl Halbwörter freigehalten oder die maximal mögliche Anzahl von 256 angegeben werden.

Während des Ablaufs des Programms kann das Indexbasisregister X mit Hilfe des Befehls ZI oder BCI auf einen anderen Wert gesetzt werden. Alle Indexadressen beziehen sich auf den jeweils durch das Register X angegebenen Indexbereich. Näheres dazu ist dem Abschnitt "Adressenrechnung" zu entnehmen.

Das Unterprogrammregister U wird benötigt, wenn Unterprogrammssprünge mit dem Befehl SU oder SUE gemacht werden. Das Unterprogrammregister U gibt dann an, in welcher Indexzelle die Rückkehradresse abgespeichert ist. Einzelheiten dazu sind dem Abschnitt "Programmteile" 3. zu entnehmen. Der Pseudobefehl zum Vorbesetzen des Registers U hat die Form

`UNTPR i` i: Indexadresse

Wird der Befehl SU oder SUE im Programm nicht verwendet, so ist das Register U zweckmäßigerweise auf den Wert 254 zu setzen. Wird dann trotzdem der Befehl SU oder SUE verwendet, so wird gemäß der Wirkung dieses Befehls ein U-Alarm gegeben.

Für den Fall, daß beim Lauf des Operators ein Fehler auftritt, muß mit dem Pseudobefehl

`ALARM m` m: Adresse eines Befehls,
bei dem ein Fehlerprogramm
beginnt.

eine Fehleradresse angegeben werden.

TAS - Handbuch

Tritt während des Laufs ein Fehler (Alarm) auf, so wird auf die angegebene Adresse gesprungen. Einzelheiten dazu sind dem Abschnitt "Sonstiges" zu entnehmen.

Für den Start eines Operators ist es erforderlich, anzugeben bei welcher Adresse der Operatorlauf gestartet werden soll. Dies wird mit dem Pseudobefehl

START adr adr: Adresse, bei der der Operator gestartet wird

angegeben. Die Adresse muß im gleichen Quellenprogramm definiert sein. Bezieht er sich auf ein anderes Quellenprogramm, so muß mit dem Pseudobefehl EXTERN die Verbindung hergestellt werden.

Für die Speicherplätze, die mit den Pseudobefehlen ASP und DSP reserviert sind, muß angegeben werden, mit welchem Wert sie vorbesetzt werden sollen. Hierzu dient der Pseudobefehl

VORBES (t, 's') t: Typenkennung
s: 12 Sedezimalzeichen

Hierbei wird so vorgegangen, daß alle benötigten Speicherplätze mit der angegebenen Typenkennung auf den angegebenen Wert gesetzt und daß dann die Speicherplätze mit der bedeutsamen Information überschrieben werden.

Die mit ASP und DSP freigehaltenen Speicherplätze haben keine bedeutsame Information, werden also nicht verändert und bleiben auf dem vorbesetzten Wert.

Diese allgemeine Vorbesetzung kann für das mit dem Pseudobefehl GEBIET definierte Gebiet überschrieben werden (siehe Abschnitt 3.4.).

Die vorstehend beschriebenen Angaben dürfen für den Operator nur einmal festgelegt werden. Enthält ein Operator mehrere Quellenprogramme, so dürfen die Anweisungen nur einmal in einem Quellenprogramm definiert werden. Eine Ausnahme bildet der Pseudobefehl STRUKT, der in jedem Quellenprogramm einmal vorkommen darf.

```
VOR= SEGM,
      XBasis XB,
XB=  ASP 30,
     UNTPR 5,
     ALARM AADR,
     VORBES (1, '000000000000'),
     START ANF,
     STRUKT (1, KONTR0LLRL),
     .
     .
     .
```

30 Indexzellen reservieren
Unterprogrammregister auf 5 setzen
Bei Fehlern wird auf die Adresse AADR gesprungen
Gestartet wird bei der Adresse ANF
Vorbesetzt wird mit 0 und Typenkennung 1

Bild 4.1 Beispiel für die Voreinstellung

Die im Pseudobefehl STRUKT mitgegebene Information wird nur in Testversionen im Alarmfall bei der Rückverfolgung der Unterprogrammaufrufhierarchie ausgewertet.

STRUKT (typ, name) typ: Zahl 1...15 oder eine Tetrade
≠ 0
name: Kontrollblockname

Der Typ gibt an, nach welchen Unterprogrammkonventionen das Quellenprogramm aufgebaut ist. Typ 1 bedeutet, daß das Quellenprogramm ein Unterprogramm nach den allgemeinen Unterprogrammkonventionen ist und daß der Kontrollblock bei der angegebenen Adresse liegt (siehe Abschnitt D, Programmteile). Der Parameter Typ=1 darf fehlen. Der Assembler gibt dann aber eine Fehlermeldung aus. Ist der Typ≠1, so darf man auf den Kontrollblocknamen verzichten.

Man kann sich die Arbeit sparen, die Pseudobefehle für die Voreinstellung zu schreiben, wenn man das Makro R&RAHMEN benutzt. Zweckmäßigerweise ruft man das Makro R&RAHMEN implizit mit der Spezifikation SPRACHE = TASR im UEBERSETZE-Kommando auf. Die Wirkung des Aufrufs ist:

- Ein Speicherbereich für 256 Indexzellen wird reserviert und ein Pseudobefehl XBasis mit der Anfangsadresse dieses Bereichs erzeugt.
- Ein Pseudobefehl UNTPR 247 wird erzeugt.
- Ein Pseudobefehl VORBES(3, 'FFFFFFFF') wird erzeugt.
- Ein Pseudobefehl ALARM zur Behandlung von Alarmen und SSR-Fehlern wird erzeugt, die Montageobjekte S&CC und S&SRF angeschlossen und die erforderlichen Anfangsaufrufe durchgeführt.
- Ein Pseudobefehl START wird erzeugt.

Wenn das Makro R&RAHMEN benutzt wird, dürfen die Pseudobefehle XBasis, UNTPR, VORBES, ALARM und START nicht mehr geschrieben werden.

Näheres zum Makro R&RAHMEN und den Wirkungen ist dem Abschnitt D 10, Standardrahmen für TAS-Programme zu entnehmen.

4.2. Programmanfang

Ist der erste Pseudobefehl SEGM benannt, so erhält das Quellenprogramm diesen Namen. Besitzt dieser Pseudobefehl mehrere Namen, dann wird sein erster Name der Quellenprogrammname. Wenn der erste Pseudobefehl SEGM nicht benannt ist, erhält das Quellenprogramm keinen Namen. In der Kopfzeile des Übersetzungsprotokolls bleibt die Position "Programmname" unbesetzt.

Der Name des Quellenprogramms wird der Name des erzeugten Montageobjekts, falls im UEBERSETZE-Kommando kein anderer Montageobjektname explizit vereinbart wird (siehe Kommandohandbuch). Bei einem namenlosen Quellenprogramm wird der Standardname STDHP verwendet, wenn im UEBERSETZE-Kommando statt eines Namens die Spezifikation -STD- angegeben wird.

4.3. Programmende

Das Ende eines Programms wird dem Assembler durch den Pseudobefehl

ENDE

angezeigt. Er ist das Zeichen für den Assembler, daß die Übersetzung beendet ist. Er zeigt also das statische Ende eines Quellenprogramms an. Damit ist er der letzte Befehl des Quellenprogramms.

Das dynamische Ende eines Operators (Ende des Operatorlaufs) wird dem Abwickler durch den SSR 0 12 mitgeteilt. Weitere Einzelheiten sind in der Unterlagensammlung "Systemdienste BS 3/1" aufgeführt.

Beim Erreichen des Pseudobefehls ENDE müssen alle mit F spezifizierten Konstanten abgeschlossen sein. Offene F-Konstanten werden abgeschlossen und eine Fehlermeldung wird ausgegeben.

4.4. Segmente

Es besteht die Möglichkeit, den Geltungsbereich von Namen, die für symbolische Adressen stehen, auf einen Teil des Quellenprogramms zu begrenzen. Zu diesem Zweck wird das Quellenprogramm in Segmente gegliedert. Dies geschieht mit Hilfe des Pseudobefehls

name = SEGM name: Name den das Segment bekommt.

Mit dem Pseudobefehl SEGM beginnt ein neues Segment. Es enthält den angegebenen Namen. Alle bisher festgelegten lokalen Namen verlieren ihre Gültigkeit. Es können unabhängig von den in anderen Segmenten verwendeten Namen neue lokale Namen festgelegt werden. Das heißt, daß lokale Namen nur in dem Segment gelten, in dem sie festgelegt wurden. Ein neuer SEGM-Befehl schließt das vorhergehende Segment ab. (Über den Geltungsbereich von Namen siehe Abschnitt 6.)

4.5. Informationseinheiten

Das Quellenprogramm besteht aus einer Folge von Informationseinheiten. Diese Informationseinheiten können enthalten

- Befehle
- Pseudobefehle
- Konstanten

Unter "Befehle" sind hier die Maschinenbefehle zu verstehen, die eins zu eins in die interne Form übersetzt werden. Zu den Pseudobefehlen gehören auch die Makroaufrufe.

Die Gliederung der Informationseinheiten ist im Abschnitt 5. ausführlich behandelt.

5. INFORMATIONSEINHEITEN

Die Informationseinheiten sind die Bausteine eines Quellenprogramms. Sie werden in der Reihenfolge der Niederschrift vom Assembler übersetzt. Die letzte Informationseinheit eines Quellenprogramms ist stets der Pseudobefehl ENDE.

Informationseinheiten können Befehle enthalten, die vom Assembler in die maschineninterne Form übersetzt werden. Des Weiteren können sie Pseudobefehle enthalten, die Anweisungen an den Assembler sind. In diesem Sinne werden auch Makroaufrufe (siehe Abschnitt "Programmenteile") als Pseudobefehle betrachtet. Als weitere Möglichkeit können sie Konstanten enthalten; sie werden in die maschineninterne Form übersetzt.

Die Informationseinheiten sind gegliedert in

- Benennung; sie gibt der Informationseinheit den symbolischen Namen
- Informationsteil; er enthält den Befehl, Pseudobefehl oder die Konstante und ist der Kern der Informationseinheit
- IE-Spezifikation; sie gibt dem Assembler an, daß der Informationsteil auf eine bestimmte Art zu behandeln ist.
- Abgrenzungsteil; er schließt die Informationseinheit ab und besteht im einfachsten Fall aus einem Komma, kann aber auch aus einem Kommentar oder einer Überschrift bestehen.

Der Abgrenzungsteil muß stets vorhanden sein; alle anderen Teile können entfallen. Damit kann eine Informationseinheit lediglich aus dem Abgrenzungsteil bestehen.

Wieweit der Spezifikationsteil und der Benennungsteil vorhanden sein müssen, vorhanden sein können oder nicht vorhanden sein dürfen, wird durch den Informationsteil bestimmt. Dabei gibt es für die Befehle und Konstanten einheitliche Regeln. Dagegen ist es für jeden Pseudobefehl speziell festgelegt.

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

In den nachfolgenden Abschnitten wird zuerst auf die drei möglichen Arten des Informationsteils eingegangen. Danach werden die anderen Teile der Informationseinheit behandelt.

5.1. Befehle

Wenn hier von Befehlen die Rede ist, so sind die Maschinen-Befehle gemeint, die also vom Assembler eins zu eins in die maschineninterne Form übersetzt werden.

Die Form der Befehle ist durch das "Befehls-Lexikon" und durch die "Große Befehlsliste" festgelegt und im Abschnitt "Befehle und Adressierung" beschrieben.

Jeder Befehl kann einen Benennungsteil haben, der dem Befehl einen symbolischen Namen gibt (siehe Abschnitt 5.4.).

Jeder Befehl kann einen Spezifikationsteil haben (siehe Abschnitt 5.5.). Erlaubt sind jedoch nur folgende IE-Spezifikationen.

K, V, B, D: Ablagebereich

0, 1, 2, 3: Typenkennung

G, U: Gerade oder ungerade Adresse

Die anderen Spezifikationen sind nicht erlaubt.

Jeder Befehl muß einen Abgrenzungsteil erhalten, der im einfachsten Fall aus einem Komma besteht.

ZX	10	X1,	
B	ANTON,		
ANF=	XBA	100,	} mit Benennung
SCHL=	HXP	-2 X5,	
BA	VSB	--V-BLOCK--	} mit Kommentar
ZI	XB1	--X.BASIS 1--	
BA	'--'/VGI,		} mit Spezifikation
XBA	100/V1,		

Bild 5.1 Beispiele für Befehle

5.2. Pseudobefehle

Pseudobefehle sind Anweisungen an den Assembler. Welche Pseudobefehle es gibt und in welcher Weise sie auf die Übersetzung einwirken, ist im Abschnitt "Konstanten, Pseudobefehle" beschrieben.

In bezug auf Benennungsteil und Spezifikationsteil hat jeder Pseudobefehl eine spezielle Festlegung. Der Abgrenzungsteil ist für jeden Pseudobefehl zwingend.

Zu den Pseudobefehlen werden auch die Makroaufrufe gezählt. Die Form des Makroaufrufs wird bei der Definition des Makros festgelegt. Es gibt Makros, die bereits definiert und im Assembler vorhanden sind und Makros, die vom Benutzer selbst definiert werden.

```

        ENDE,
        WEND,
ANF=   ASP  2K8,
DB=    ASP  100/D,
        EXTERN  ZMO(A,B,C),

```

Bild.5.2 Beispiele für Pseudobefehle

5.3. Konstanten

Bei den Konstanten unterscheiden wir

- Gleitkommakonstanten
- Gleitkommakonstanten mit doppelter Genauigkeit
- Festkommakonstanten
- Tetradenkonstanten
- Oktadenkonstanten
- Textkonstanten
- Bitfeldkonstanten
- Adressenkonstanten

Sie werden vom Assembler in die interne Form übersetzt.

Konstanten können einen Benennungsteil haben, der der Konstanten einen symbolischen Namen gibt (siehe Abschnitt 5.4.) und das erste Wort der Konstanten adressiert.

```

NR=   375/V      --ANF.WERT--
      0.128,
      13.6E5/V,
      1.85D13,
      'TELEFUNKEN' / 0,
PI=   3.14E,
      'A3F 015 4FA',
A=    3/H3,
DB=   ANTON/AV1,

```

Bild 5.3 Beispiele für Konstanten

Die Konstanten können auch einen Spezifikationsteil haben. Die erlaubten Spezifikationen sind bei den einzelnen Konstanten verschieden und werden jeweils angegeben.

Jede Konstante muß einen Abgrenzungsteil haben, der im einfachsten Fall aus einem Komma besteht.

Die Konstanten sind im Abschnitt "Konstanten, Pseudobefehle" beschrieben.

5.4. Benennung

Befehle und Konstanten können mit einem Namen versehen werden. Unter diesem Namen können sie aufgerufen werden. Dies trifft auch für einige Pseudobefehle zu. Bei einigen jedoch ist die Angabe eines Namens unbedingt erforderlich oder nicht zugelassen. Dies ist jeweils beim Pseudobefehl vermerkt.

Ein Name beginnt stets mit einem Buchstaben. Es können Buchstaben und Ziffern in beliebiger Reihenfolge folgen. Leertasten und Sonderzeichen (auch der Gedankenstrich oder Punkt) sind nicht erlaubt. Abgeschlossen wird der Name durch das Gleichheitszeichen. Zwischen Namen und Gleichheitszeichen können beliebig Leertasten eingeschoben werden.

Der Geltungsbereich eines Namens kann auf ein Segment begrenzt sein (lokale Namen) oder für alle Segmente des Quellenprogramms (globale Namen). Lokale Namen werden nicht gekennzeichnet. Bei globalen Namen muß jedoch zwischen Namen und Gleichheitszeichen ein Punkt eingeschoben werden.

name = lokaler Name

name . = globaler Name

Wird in einem Segment ein lokaler Name festgelegt, der bereits als globaler Name festgelegt wurde, so gilt in diesem Segment der lokale Name. Punkt und Gleichheitszeichen sind nicht Bestandteil des Namens sondern Steuerzeichen für den Assembler.

Einem Befehl, einer Konstanten oder einem Pseudobefehl (bei dem eine Benennung erlaubt ist) können beliebig viele Namen gegeben werden. In diesem Fall wird hinter dem abschließenden Gleichheitszeichen des ersten Namens der zweite Name angegeben usw.

name1 = name2 = name3. =

Ein Aufruf kann unter allen angegebenen Namen erfolgen. Damit ist es z.B. möglich, einem Namen einen globalen und einem weiteren Namen einen lokalen Geltungsbereich zu geben.

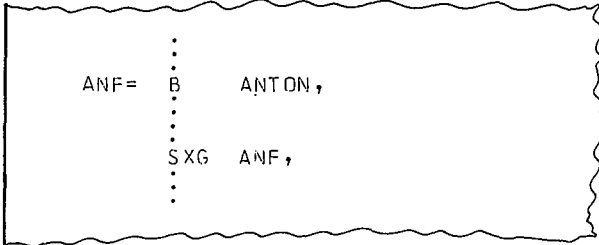


Bild 5.4 Beispiel für Benennung bei Befehlen

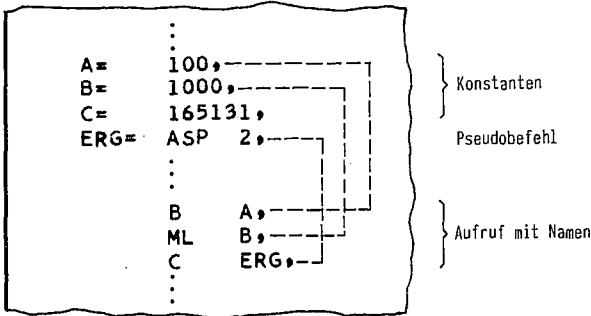


Bild 5.5 Beispiel für Benennung

5.5.1. Ablagespezifikation

Diese Spezifikation gibt an, in welchem Ablagebereich der Befehl oder die Konstante abgelegt werden soll. Hierbei bedeuten:

- K = Konstantenbereich
- V = Variablenbereich
- B = Befehlsbereich
- D = Datenbereich

Diese Spezifikation darf auch bei den Pseudobefehlen ASP und DSP verwendet werden.

Selbstverständlich kann jeweils nur einer dieser Buchstaben angegeben werden.

Wird eine implizit bereits gegebene Ablagespezifikation explizit angegeben, so kann sie in bezug auf den Pseudobefehl STARR von Bedeutung sein, da sie die Ablagespezifikationen des Pseudobefehls STARR überschreibt.

5.5. Spezifikationen

Die hier beschriebenen Spezifikationen dürfen nicht mit denen verwechselt werden, die gemäß "Große Befehlsliste" und "Befehls-Lexikon" im Adressenteil des Befehls stehen.

Bei der Niederschrift von Befehlen und Konstanten ist implizit eine Ablage enthalten. Das heißt, ohne spezielle Angaben werden sie in bezug auf

- Adressenzone
- Typenkennung
- rechts- oder linksbündig
- usw.

in einer bestimmten Art abgelegt. Bei den Befehlen gilt, daß sie im B-Bereich mit der Typenkennung 2 in einem Halbwort abgelegt werden. Bei den Konstanten ist jeweils vermerkt, welche implizite IE-Spezifikation sie haben.

Wird eine andere Ablage gewünscht, als die implizit vorgegebene, so kann dies explizit durch die Angabe einer Spezifikation hinter dem Befehl oder hinter der Konstanten und hiervon durch einen Schrägstrich getrennt, erfolgen.

.../s

Die Spezifikationen werden mit einem Buchstaben oder einer Zahl angegeben und können in beliebiger Reihenfolge stehen. Befehle, deren Adressenteil ein Literal enthält, dürfen nicht mit einer Spezifikation versehen werden.

5.5.2. Typenkennung

Diese Spezifikation gibt an, mit welcher Typenkennung der Befehl oder die Konstante abzulegen ist. Es sind die Ziffern 0, 1, 2 oder 3 erlaubt. Die Spezifikation bewirkt, daß das Ganzwort, in dem der Befehl oder die Konstante abgelegt wird, die angegebene Typenkennung erhält.

Werden zwei Befehle oder Halbwort- und Adressenkonstanten mit verschiedenen Typenkennungen in einem Ganzwort abgelegt, so wird nur die Typenkennungsspezifikation des Halbwortes mit der geraden Ablageadresse wirksam. Die Typenkennung richtet sich also immer nach dem linken Halbwort, auch dann, wenn das rechte Halbwort eine explizite Typenkennungsangabe besitzt.

Es ist also zweckmäßig, bei Halbwörtern, die in einem Wort mit bestimmter Typenkennung untergebracht werden sollen, gleichzeitig mit der Typenkennung die Spezifikation G anzugeben. Es ist damit auf jeden Fall gewährleistet, daß das Ganzwort, in dem das Halbwort steht, die geforderte Typenkennung besitzt. Ist nicht die Spezifikation G, sondern U gewünscht, so ist es zweckmäßig, ein weiteres Halbwort zu belegen, z. B. mit NULL.

5.5.3. Halbwort

Die mit der Spezifikation H versehene Konstante wird in einem Halbwort abgelegt. Sie ist sinngemäß nur bei Festkomma-, Tetraden-, Oktaden- und Textkonstanten erlaubt. Die Konstanten dürfen dabei den Wert nicht überschreiten, der für ein Halbwort möglich ist.

Wird zusätzlich eine spezielle Typenkennung angegeben, so gilt das im Abschnitt 5.5.2. Gesagte.

Bei Halbwörtern muß darauf geachtet werden, daß sie eine andere Typenkennung bekommen können als man nach der Niederschrift erwartet (siehe Abschnitt 5.5.2.).

5.5.4. Gradzahlige oder ungradzahlige

Bei Befehlen und Konstanten, die ein Halbwort belegen, kann angegeben werden, ob sie in der linken Hälfte eines Ganzworts (gradzahlige Adresse) oder in der rechten Hälfte eines Ganzworts (ungradzahlige Adresse) abgelegt werden sollen.

G = gradzahlige Adresse
U = ungradzahlige Adresse

Erfüllt das nächste freie Halbwort nicht diese Bedingung, so wird es auf Null gesetzt (bei Oktaden ist dies die Oktade 0, bei Befehlen der NULL-Befehl) und das darauffolgende Halbwort genommen.

5.5.5. Links- oder rechtsbündig

Tetradenkonstanten werden, wenn sie kürzer als ein Ganzwort sind (oder bei Spezifikation H kürzer als ein Halbwort) rechtsbündig abgelegt und der Rest mit Nullen aufgefüllt. Sollen sie jedoch linksbündig abgelegt werden, so kann dies durch die Spezifikation L erreicht werden.

Oktaden und Textkonstanten werden linksbündig, in einem Wort beginnend, abgelegt. Sollen sie rechtsbündig abgelegt werden, so ist das mit der Spezifikation R möglich. Dies gilt auch, wenn diese Konstanten länger als ein Ganzwort sind. Es wird dann das erste Wort links mit Nullen (Oktade 0) aufgefüllt, so daß die letzte Oktade im letzten Wort ganz rechts steht. Ist die Spezifikation H angegeben, so dürfen diese Konstanten nicht länger als ein Halbwort sein. Sind sie kürzer, so kann mit der Spezifikation R die Konstante ebenfalls rechtsbündig abgelegt werden. Die Spezifikation F ist nicht gleichzeitig erlaubt.

5.5.6. Fortsetzung

Oktaden- und Textkonstanten können länger als ein Ganzwort sein. Es werden aufeinanderfolgende Ganzwörter aufgefüllt. Das letzte Ganzwort wird im Normalfall mit Null (Oktade 0) aufgefüllt.

5.5.7. Markieren (Bit 1 oder Bit 2)

Die Spezifikation M bewirkt bei Festkomma- und Gleitkommakonstanten, daß das 1. Bit (linkes Bit) des Ganzworts gesetzt wird. Damit sind diese Zahlen markiert. Bei einer Festkommazahl halber Wortlänge ist die Spezifikation M nicht erlaubt. Bei Gleitkommazahlen doppelter Wortlänge wird nur das 1. Bit des ersten Ganzworts gesetzt.

Bei Adressenkonstanten wird das erste Bit des Halbworts gesetzt. Bei den anderen Konstanten ist die Spezifikation M nicht erlaubt.

Die Spezifikation N ist nur bei Adressenkonstanten erlaubt und setzt das zweite Bit (zweite von links) des Halbworts.

5.5.8. Adressenkonstante

Die Spezifikation A wird nur bei Adressenkonstanten verwendet. Sie ist zur Kennzeichnung einer Adressenkonstanten unbedingt erforderlich.

```

KONST=100/H3,
    BA  0/V,
    '3AF'/LV3,
    .361/M,
    ''TEXT''/R2,
LEERT='' ''/HR1,
    B   ANTON/G,
    120/H2U,
    120/H2G,0/H,
ADR=  DBEREICH/A,
XBAS= X/AG,0/H,
A=    SUMME/AMNO,
OKT=  ''TELEFUNKEN'',
      ''(114, ''130'')'',
      ''BIS'',
      ''(114, ''140'')'',

```

Bild 5.6 Beispiel für Spezifikationen

5.7. Kommentar

Ein Kommentar besteht aus einem Text, der beidseitig durch je zwei Minuszeichen (Gedankenstriche) eingeschlossen wird. Im Text sind alle Zeichen erlaubt, die im Eingabecode möglich sind. Da im allgemeinen die Kommentare auch im Protokoll erscheinen, sollten die Zeichen auch im Code des Ausgabemediums vorhanden sein, auf dem das Protokoll erstellt wird.

Kommentare dienen zur Programmierläuterung und sind keine Information für den Assembler; sie werden unverändert ins Übersetzungsprotokoll übernommen. Die Einschließungszeichen werden dabei mit protokolliert.

Nachstehende Zeichen sind im Text jedoch nicht erlaubt:

- zwei aufeinanderfolgende Minuszeichen,
- ein Minuszeichen am Anfang oder Ende des Textes,
- die Zeichenkombination *(), da sie für einen Ersetzungsbezug gehalten werden kann.

Ein Kommentar kann an Stelle des Kommas als Abgrenzungsteil stehen. Er wird dann im Protokoll hinter der Informationseinheit in der gleichen Zeile gedruckt. Steht er hinter einem Komma, so ist er eine selbständige Informationseinheit und wird im Übersetzungsprotokoll ganz links in eigener Zeile gedruckt.

```

SXN  ANF      --LETZTER DURCHLAUF--
BNZ  X1  X2,  --NAECHSTES ZEICHEN--

```

Bild 5.7 Beispiel für Kommentar

5.6. Abgrenzungsteil

Der Abgrenzungsteil beendet eine Informationseinheit und besteht im allgemeinen aus dem Zeichen Komma (,). Dieses Komma gibt dem Assembler an, daß ein Befehl, Pseudobefehl oder eine Konstante (ggf. mit Spezifikation) beendet ist. Der Abgrenzungsteil ist also immer erforderlich.

An Stelle eines Kommas kann auch ein Kommentar oder eine Überschrift stehen. Dies ist in den Abschnitten 5.7. und 5.8. erläutert.

Es können mehrere Abgrenzungsteile aufeinanderfolgen, also auch mehrere Kommas. Dies bewirkt, daß im Protokoll in einer Zeile nur das Komma steht, da diese Informationseinheit nur aus einem Komma besteht.

5.8. Überschrift

Überschriften dienen, wie die Kommentare, zur Erläuterung der Programme. Für sie gelten die gleichen Bedingungen wie für Kommentare. Gegenüber den Kommentaren haben sie jedoch folgende Besonderheiten:

Die Überschrift beginnt mit drei Minuszeichen. Im Übersetzungsprotokoll wird bei jeder Überschrift ein neues Blatt begonnen und die Überschrift an den Kopf dieses und der folgenden Blätter gesetzt.

Eine neue Überschrift bewirkt wiederum einen Blattwechsel, und dieses neue Blatt und alle folgenden werden mit der neuen Überschrift versehen.

6. GELTUNGSBEREICH VON NAMEN

Namen, die in Quellenprogrammen benutzt werden, können auf verschiedene Art festgelegt werden oder sind bereits festgelegt. Entsprechend ihrer Festlegung haben sie auch einen verschiedenen Gültigkeitsbereich.

Namen für Befehle sind im "Befehls-Lexikon" bzw. in der "Großen Befehlsliste" festgelegt. Die Namen der Pseudobefehle, soweit sie keine Makroaufrufe sind, sind durch den TAS-Assembler festgelegt. Auf diese Namen braucht nicht weiter eingegangen zu werden.

Festgelegte Namen können innerhalb ihres Geltungsbereichs in Adressenteilen von Befehlen verwendet werden.

Die Länge der Namen ist auf 31 Zeichen begrenzt.

Ausnahmen bilden Montage-, Kontakt- und Zonenamen, von denen nur die ersten 12 Zeichen zur Unterscheidung herangezogen werden. Bei Gebiets- und Lückennamen wirken nur die ersten 6 Zeichen unterscheidend. Von dem Programmnamen werden beim UEBERSETZE-Kommando nur die ersten 12 Zeichen zur Unterscheidung verwendet. Innerhalb des Quellenprogramms gelten alle Zeichen.

6.1. Lokale Namen

Lokale Namen gelten nur innerhalb des Segments, in dem sie festgelegt werden. Sie werden stets im Benennungsteil einer Informationseinheit, also links von einem Befehl, einer Konstanten oder einem Pseudobefehl (soweit erlaubt) festgelegt. Lokale Namen sind dadurch gekennzeichnet, daß sie mit einem Gleichheitszeichen abgeschlossen sind.

```
name =                name: lokaler Name
```

Ein Pseudobefehl SEGM hebt die Gültigkeit aller bisher festgelegten lokalen Namen auf. Selbstverständlich gelten lokale Namen nur im eigenen Quellenprogramm.

Indexnamen, die mit dem Pseudobefehl INDEX definiert werden und nicht mit einem Punkt abgeschlossen sind, haben ebenfalls einen lokalen Geltungsbereich.

6.2. Globale Namen

Globale Namen gelten im gesamten Quellenprogramm. Sie werden ebenfalls, wie die lokalen Namen, im Benennungsteil von Befehlen, Konstanten und Pseudobefehlen (soweit erlaubt) festgelegt. Zur Unterscheidung von den

lokalen Namen werden sie jedoch durch Punkt und Gleichheitszeichen abgeschlossen.

```
name . =              name: globaler Name
```

Der Punkt ist dabei nicht Bestandteil des Namens.

Wird in einem Segment ein lokaler Name verwendet, der auch als globaler Name in einem anderen Segment vorkommt, so gilt in diesem Segment der lokale Name.

In den Bildern 6.1 und 6.2 ist in einem Beispiel der Geltungsbereich von globalen und lokalen Namen gezeigt.

Indexnamen, die implizit definiert werden, haben stets einen globalen Geltungsbereich. Werden Indexnamen explizit durch den Pseudobefehl INDEX definiert, so müssen globale Namen durch einen Punkt abgeschlossen werden.

6.3. Montagenamen

Montagenamen sind Namen, die auch nach der Übersetzung als Namen bestehen bleiben und nicht in eine Adresse übersetzt werden. Die Montagenamen werden vom Montageprogramm benötigt, um die verschiedenen Montageobjekte zu einem Operator zusammenfügen zu können.

Durch das UEBERSETZE-Kommando wird ein Quellenprogramm in den Montagecode übersetzt und wird damit zum Montageobjekt. Dieses Montageobjekt erhält einen Montagenamen, der im UEBERSETZE-Kommando angegeben ist.

In jedem Quellenprogramm können mit Hilfe des Pseudobefehls

```
.EINGG mon           mon: Montagenamen des eigenen Quellen-  
                    programms
```

weitere Montagenamen vereinbart werden.

Der Name, der mit Hilfe dieses Befehls zum Montagenamen wird, muß im Programm definiert sein; er muß also ein Wort adressieren. Andere Quellenprogramme können sich auf diese Montagenamen beziehen.

Will ein Quellenprogramm sich auf einen Namen in einem anderen Quellenprogramm beziehen, so kann es dies nur über die Montagenamen. Zu diesem Zweck muß mit Hilfe des Pseudobefehls

Benennung:	Lokal Global	Benennungs- Segment	Gültig im Segment				
			1	2	3	4	5
DOKUMENTATION	L	1	x				
A	L	1	x				
TEXT	L	1	x				
T2	G	1	x	x		x	x
X1	L	1	x				
X2	G	1	x	x	x	x	x
X3	L	1	x				
RECHT	L	2		x			
K3	L	2		x			
A	L	2		x			
MEDIZIN	L	3			x		
T2	L	3			x		
A	G	3			x	x	
HSP	G	3	x	x	x	x	x
XA	L	3			x		
XB	L	3			x		
XC	G	3	x	x	x	x	x
TECHNIK	G	4	x	x	x	x	x
TEXT	G	4		x	x	x	x
K3	L	4				x	
ASTRONOMIE	G	5	x	x	x	x	x
K3	G	5	x		x		
KONST	G	5	x	x	x	x	x
A	L	5					x
K3	L	5					x
K4	L	5					x

Bild 6.2 Liste zum Beispiel Bild 6.1

EXTERN mon

mon: Montagenamen des fremden Quellenprogramms

angegeben werden, auf welchen Montagenamen eines fremden Programms sich das eigene Quellenprogramm beziehen will. Dieser Pseudobefehl bedeutet einmal, daß dieser Name im eigenen Programm nicht definiert und zum anderen, daß bei der Montage das Quellenprogramm, zu dem dieser Montagenamen gehört, auf jeden Fall montiert wird. Es braucht also beim MONTIERE-Kommando nicht speziell die Montage dieses fremden Quellenprogramms angegeben zu werden.

In den Bildern 6.3 und 6.4 sind im Zusammenhang mit Kontaktnamen (siehe nächsten Abschnitt) Beispiele angegeben.

Mehrere Montageobjekte werden durch das MONTIERE-Kommando zu einem Operator montiert. Im Kommando werden dabei die Montagenamen der Montageobjekte angegeben.

Beim UEBERSETZE- und MONTIERE-Kommando werden nur die ersten 12 Zeichen des Montagenamens zur Unterscheidung verwendet.

6.4. Kontaktnamen

Kontaktnamen haben eine ähnliche Bedeutung wie die Montagenamen; sie können jedoch nur im Zusammen-

hang mit Montagenamen verwendet werden. Damit kann die Anzahl der Montagenamen herabgesetzt werden, die ja über mehrere Quellenprogramme eindeutig sein müssen.

In jedem Quellenprogramm ist bereits ein Montagenamen vorhanden, nämlich der Name, den das Montageobjekt mit dem UEBERSETZE-Kommando erhält. Bereits in Verbindung mit diesem Namen können Kontaktnamen verwendet werden.

Kontaktnamen werden mit Hilfe des Pseudobefehls

EINGG (k)

k: Kontaktnamen (mehrere Kontaktnamen durch Komma trennen)

festgelegt. Diese Kontaktnamen müssen als Namen im Quellenprogramm definiert sein; sie müssen ein Wort adressieren.

Will sich ein Quellenprogramm auf den Namen eines anderen Quellenprogramms beziehen, der dort als Kontaktnamen festgelegt wurde, so kann es das nur über einen der Montagenamen des fremden Quellenprogramms. Zu diesem Zweck muß mit Hilfe des Pseudobefehls

EXTERN mon (k)

mon: Montagenamen des fremden Quellenprogramms

k: Kontaktnamen des fremden Quellenprogramms (mehrere Kontaktnamen durch Komma getrennt)

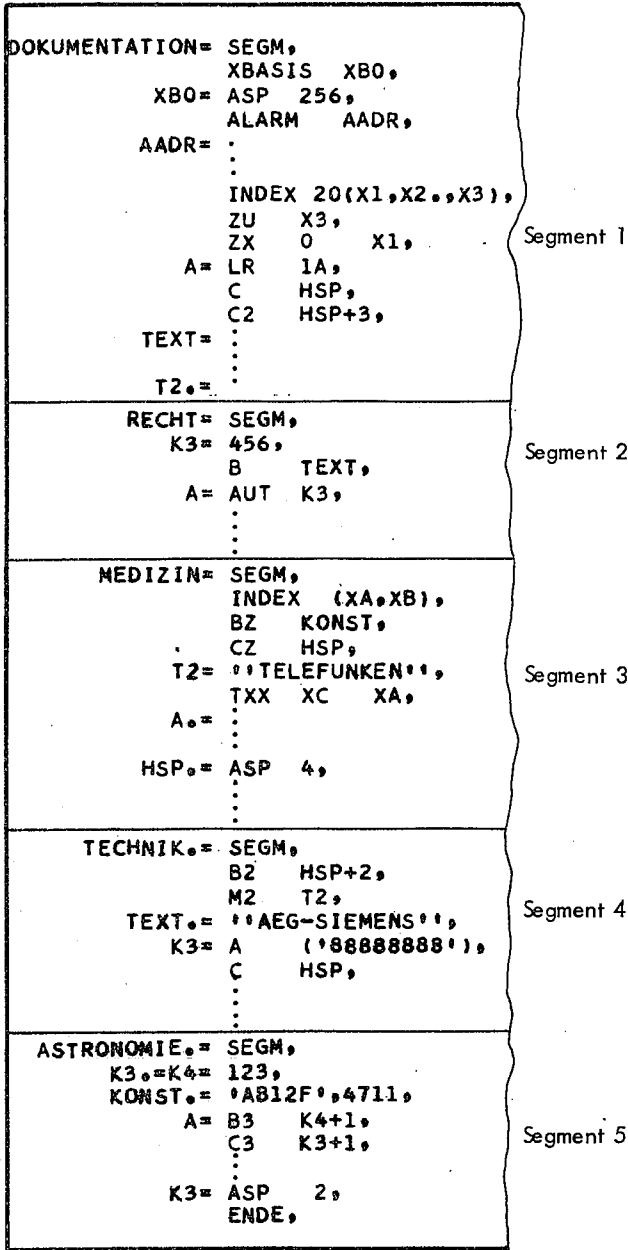


Bild 6.1 Beispiel für Geltungsbereich der Namen

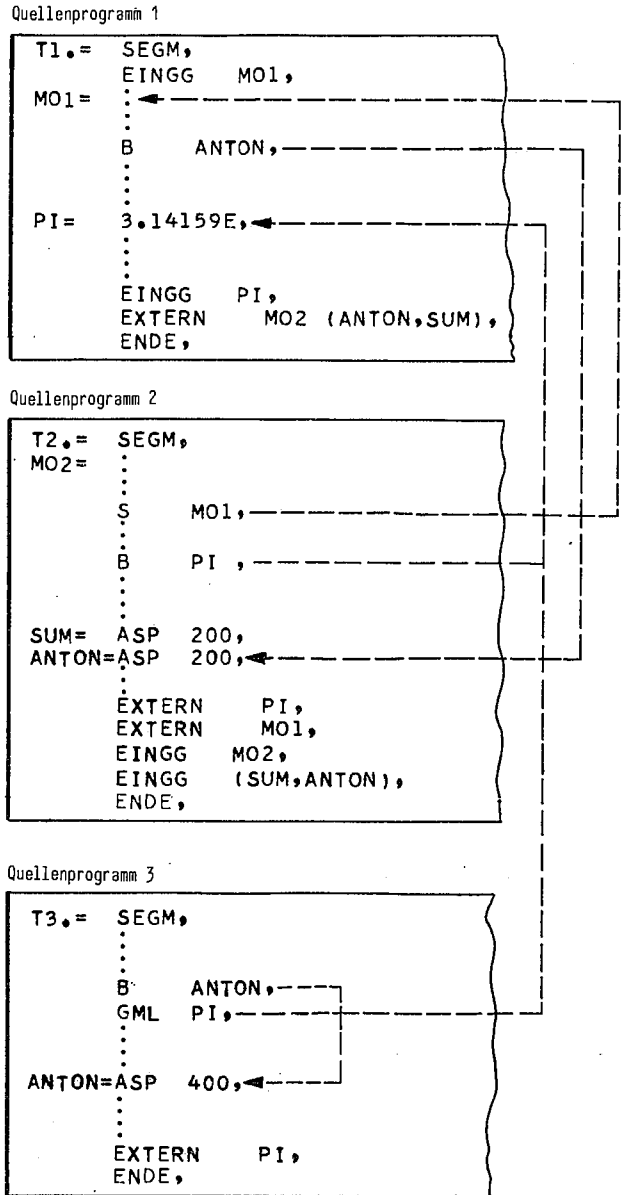


Bild 6.3 Beispiele für Pseudobefehle EXTERN und EINGG

TAS - Handbuch

angegeben werden, auf welche Montagennamen eines fremden Quellenprogramms sich das eigene Quellenprogramm beziehen will. Zusätzlich ist anzugeben, auf welche Kontaktnamen innerhalb dieses fremden Quellenprogramms Bezug genommen wird. Die weitere Bedeutung dieses Befehls ist im Abschnitt 6.3. beschrieben.

Wird von einem Quellenprogramm auf ein anderes Quellenprogramm Bezug genommen, so bedarf es natürlich einer Abmachung zwischen den Quellenprogrammen, welche Montage- und welche Kontaktnamen verwendet werden. In den Bildern 6.3 und 6.4 sind Beispiele gezeigt.

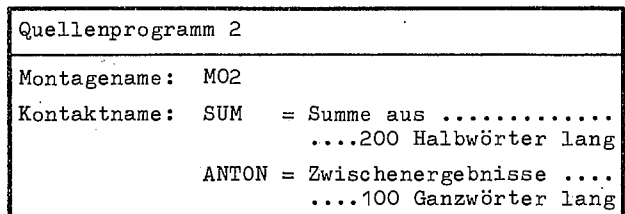
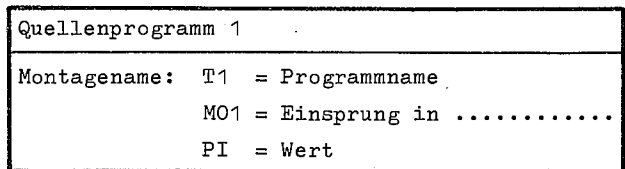


Bild 6.4 Festlegungen in den Quellenprogrammen für Bezüge von anderen Quellenprogrammen gemäß Beispiel in Bild 6.3

6.5. Globale Makronamen

Hier wird die Kenntnis der Makrotechnik (siehe Abschnitt "Programmteile") vorausgesetzt. Einige Makros sind bereits im Assembler definiert, so daß sie aufgerufen werden können, ohne daß der Benutzer irgendwelche Definitionen treffen muß. Er kann also die Makroaufrufe so benutzen, als wären sie Pseudobefehle an den Assembler.

Der für diese Makros festgelegte Name hat globale Bedeutung und darf für andere Makros nicht verwendet werden. Um dies auszuschließen, haben alle globalen Makronamen ein besonderes Zeichen in ihrem Namen. Es ist dies der Stern (*). Er darf in Namen, die vom Benutzer festgelegt werden, nicht verwendet werden.

Die Wirkung dieser Makroaufrufe ist der jeweiligen Beschreibung zu entnehmen, ebenso die Art, wie der Aufruf zu schreiben ist (welche aktuellen Parameter).

6.6. Globaler Makroparameter

Hier wird die Kenntnis der Makrotechnik (siehe Abschnitt "Programmteile") vorausgesetzt. Ähnlich wie bei der Definition eines Makros Namen für formale Makroparameter festgelegt werden, ist für das gesamte Quellenprogramm ein formaler Makroparameter festgelegt. Er hat den Namen

VERSION *

und kann im gesamten Quellenprogramm als formaler Parameter verwendet werden.

Der aktuelle Wert wird dem formalen Parameter VERSION * durch das UEBERSETZE-Kommando mit der Spezifikation VERSION zugewiesen. Steht also im UEBERSETZE-Kommando (unter anderem)

..., VERSION = 100, ...

so wird überall dort, wo sich der formale Parameter VERSION * befindet, bei der Übersetzung der Wert 100 eingesetzt.

Wir können in bezug auf den globalen Makroparameter folgende Analogie herstellen. Das Quellenprogramm entspricht dem Makro, das den formalen Parameter VERSION * hat. Das UEBERSETZE-Kommando entspricht dem Makroaufruf und der Wert, der der Spezifikation VERSION zugewiesen wird, ist der aktuelle Parameter des Makros (Quellenprogramms).

Der formale Parameter VERSION * kann selbstverständlich durch den Pseudobefehl FORM aufgegliedert werden. Er kann in Verbindung mit dem Pseudobefehl VERS verwendet werden, um verschiedene Versionen der Übersetzung des Quellenprogramms zu bekommen.

6.7. Quellenprogrammname

Ist der erste Pseudobefehl SEGM benannt, so erhält das Quellenprogramm diesen Namen. Besitzt dieser Pseudobefehl mehrere Namen, dann wird sein erster Name der Quellenprogrammname. Wenn der erste Pseudobefehl SEGM nicht benannt ist, erhält das Quellenprogramm keinen Namen. In der Kopfzeile des Übersetzungsprotokolls bleibt die Position "Programmname" unbesetzt.

Der Name des Quellenprogramms wird der Name des erzeugten Montageobjekts, falls im UEBERSETZE-Kommando kein anderer Montageobjektname explizit vereinbart wird (siehe Kommandohandbuch). Bei einem namenlosen Quellenprogramm wird der Standardname STDHP verwendet, wenn im UEBERSETZE-Kommando statt eines Namens die Spezifikation -STD- angegeben wird.

Von dem Programmnamen werden beim UEBERSETZE-Kommando nur die ersten 12 Zeichen zur Unterscheidung verwendet. Innerhalb des Quellenprogramms gelten alle Zeichen.

6.8. Segmentname

Die Quellenprogramme können in Segmente gegliedert werden. Dadurch besteht die Möglichkeit, den Geltungsbereich von Namen, die für symbolische Adressen stehen, auf einen Teil des Quellenprogramms zu begrenzen. Jedes Segment beginnt mit dem Pseudobefehl SEGM.

name = SEGM name: Name des Segments

Die Segmente können benannt werden. Ein neuer Pseudobefehl SEGM schließt das vorhergehende Segment ab. Lokale Namen gelten nur in dem Segment, in dem sie festgelegt wurden. Im neuen Segment gelten alle bisher festgelegten lokalen Namen nicht mehr. Dagegen sind globale Namen im gesamten Programm ansprechbar. Kommt in einem Segment ein globaler Name auch als lokaler Name vor, dann ist der Geltungsbereich des globalen Namens in diesem Segment aufgehoben.

Der Name eines Segments ist auch gleichzeitig eine symbolische Adresse für das nächste freie Halbwort in derjenigen Adressenzone, in die ein unmittelbar hinter dem Pseudobefehl SEGM stehender, nicht spezifizierter Befehl abgelegt werden würde.

Ein besonderer Fall ist zu beachten. Folgt auf einen benannten Pseudobefehl SEGM unmittelbar ein Pseudobefehl STARR, so wird im allgemeinen der Name des Segments nicht einer Informationseinheit derselben Zone zugeordnet, der die folgenden Informationseinheiten zugeordnet werden. Dadurch kann man die folgenden Informationseinheiten nicht mit dem Namen des Pseudobefehls SEGM aufrufen. Dies kann dadurch umgangen werden, daß man den Pseudobefehl STARR vor den benannten Pseudobefehl SEGM legt.

6.9. Oktadename

Oktadennamen werden in Verbindung mit Textkonstanten verwendet. Für einige Oktaden - hauptsächlich für die Steuerzeichen - sind bereits Namen festgelegt (siehe Zentralcode). Mit Hilfe des Pseudobefehls

```
name = TEXT k      name: Oktadename
                   k: Oktadenwert, dezimal oder sedezimal
```

können weitere Namen festgelegt werden. Sie dürfen die für Namen übliche Form haben und können mit anderen Namen nicht verwechselt werden, da sie nur bei Textkonstanten verwendet werden.

```
A= TEXT 224,
SZ= TEXT 'FD',
UND= TEXT 119,
```

Bild 6.5 Beispiele für Pseudobefehl TEXT

6.10. Zonenname

Zonenamen müssen innerhalb eines Operators eindeutig sein und gelten im gesamten Operator.

Die Informationseinheiten eines Quellenprogramms werden vom Assembler Adressenzonen zugeordnet. Implizit sind 13 Adressenzonen vorhanden; sie haben keinen Namen, sondern werden durch zwei Zeichen gekennzeichnet (siehe dazu Abschnitt 3.).

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

Diese Kennzeichnungen werden bei den Pseudobefehlen

```
ABLAGE zone (k)      zone: Name der Adressenzone
:                   k: Ablagekennung
                   K0 V0 B0 D0
AEND (k)             K1 V1 B1 D1
                   K2 V2 B2 D2

                   mehrere Angaben durch Komma
                   trennen
```

benötigt. Darüber hinaus können mit den Pseudobefehlen

```
name = ZONE a      name: Name der Adressenzone
name = CZONE a    a: Adressierungsbedingung
                   K, B, DK, F
name = FZONE a    V, BV, D, FV oder I
```

explizite Adressenzonen eröffnet werden. Sie erhalten den angegebenen Namen. Der hier festgelegte Zonenname kann nur beim Pseudobefehl ABLAGE verwendet werden. Eine Verwechslung mit gleichlautenden Namen, die nicht Zonenamen sind, ist nicht möglich. Zonenamen gelten im gesamten Operator.

Es dürfen mehrere Common-Zonen gleichen Namens definiert werden. Sie werden nur einmal im Adressenraum (entsprechend der längsten) angeordnet.

6.11. Gebietsnamen und Lückennamen

Bei den Gebietsnamen wird unterschieden zwischen

- operatorspezifischen Gebietsnamen
- prozeßspezifischen Gebietsnamen
- internen Gebietsnummern.

Der operatorspezifische Gebietsname kann durch die Abwicklerdienste definiert werden und gilt dann für den Operatorlauf (dynamischer Vorgang). Er kann aber auch durch den Pseudobefehl GEBIET definiert werden und gilt dann für den Operator (statischer Zustand). Der operatorspezifische Gebietsname muß innerhalb des Operators bzw. des Operatorlaufs eindeutig sein.

Der prozeßspezifische Gebietsname kann durch die Abwicklerdienste oder durch den Pseudobefehl GEBIET definiert werden und gilt dann für den Prozeß. Er muß innerhalb des Prozesses eindeutig sein.

Eine interne Gebietsnummer kann nur vom System vergeben werden und ist innerhalb des Systems eindeutig. Sie wird bei der dynamischen Erstellung eines Gebietes dem Operatorlauf mitgeteilt.

Gebietsnamen finden Verwendung bei den Abwicklerdiensten, bei denen auch die durch den Pseudobefehl GEBIET definierten Gebietsnamen verwendet werden können. Näheres dazu ist der Beschreibung der Abwicklerdienste zu entnehmen.

Des Weiteren werden Gebietsnamen bei den Pseudobefehlen GEBIET, ZONAN und GEBAN verwendet. Mit dem Pseudobefehl GEBIET wird ein Gebiet explizit vereinbart.

```
GEBIET (p)  p: Gebietsparameter
             p0: operatorspezifischer Gebietsname
             p1: prozeßspezifischer Gebietsname
```

Für jedes Gebiet können mehrere Parameter (insgesamt 10) angegeben werden. Mit zwei Parametern können Gebietsnamen vereinbart werden. Der operatorspezifische Gebietsname hat die Form

ONAME = name name: operatorspezifischer
Gebietsname

Dieser Gebietsname muß in allen zum Operator gehörenden Montageobjekten eindeutig sein. Kennzeichnend wirken nur die ersten 6 Zeichen. Der prozeßspezifische Gebietsname hat die Form

PNAME = name name: prozeßspezifischer
Gebietsname

Dieser Gebietsname muß in dem Prozeß, zu dem der Operator gehört, eindeutig sein.

Es wirken auch hier nur die ersten 6 Zeichen unterscheidend. Mindestens einer der beiden Gebietsnamen muß angegeben werden. Mit dem Pseudobefehl

ZONAN gebiet (zone)

gebiet: Name des Gebietes
zone: Name der Adressenzonen,
die dem Gebiet zugeordnet
werden sollen (mehrere
durch Komma trennen), Frei-
haltezone nicht erlaubt.

werden statisch definierten Gebieten Adressenzonen zugeordnet. Hier können nur Gebietsnamen verwendet werden, die mit dem Pseudobefehl GEBIET definiert wurden. Die mit dem Pseudobefehl GEBIET definierten Gebiete müssen mit dem Pseudobefehl

GEBAN bezug (liste)

bezug: Anfang der Gebietszuordnung im
Adressenraum;
Seitennummer (dezimal oder sedezimal)
Gebietsname oder
Lückename
liste: Gebietsname,
Lückename
Dezimalzahl oder
Sedezimalzahl
(mehrere Angaben durch Komma trennen)

im Adressenraum angeordnet werden. Bei diesem Pseudobefehl können also auch nur Gebietsnamen verwendet werden, die mit dem Pseudobefehl GEBIET definiert wurden.

Ein Lückename wird nur bei den Pseudobefehlen LUECKE, ZONAN und GEBAN verwendet. Er muß mit den operatorspezifischen Gebietsnamen innerhalb des Operators eindeutig sein. Ein Lückename darf auch nicht mit einem Gebietsnamen gleichlautend sein. Mit dem Pseudobefehl

LUECKE name a name: Name, den die Lücke erhält
max. 6 Zeichen
a: Adressierungsbedingungen
K, V, B, D, BV, DK, F oder FV
mit D vorbesetzt

werden eine Lücke und der Lückename vereinbart. Ihnen werden mit dem Pseudobefehl ZONAN Adressenzonen (Freihaltezone) zugeordnet. Mit dem Pseudobefehl GEBAN wird die Lücke im Adressenraum angeordnet, d.h. es wird Platz reserviert. Von den Gebietsnamen und Lückennamen werden intern nur die ersten 6 Zeichen verwendet.

Namen	Segment	Quellenprogramm	Montageobjekt	Operator	Prozeß
Speicher- und Indexnamen	lokal	x			
	global	x ¹⁾	x		
Segmentnamen	lokal	x			
	global	x ¹⁾	x		
Quellenprogrammnamen	x	x	x ²⁾	x ²⁾	
Montagenamen	x	x	x		
Kontaktnamen	x	x	x ³⁾		
Zonennamen	x	x	x		
Lückennamen	x	x	x	x	
Operatorspez. Gebietsnamen		x	x	x	
	Prozeßspez.	x	x	x	x

1) Kommt in einem Segment ein globaler Name auch als lokaler Name vor, dann gilt der lokale Name.
2) Wenn im UEBERSETZE- bzw. MONTIERE-Kommando kein anderer Name vereinbart wird, gilt der Quellenprogrammname.
3) Nur in Verbindung mit einem Montagenamen gültig.

Bild 6.6 Übersicht über den Geltungsbereich von Namen

7. ELEMENTE DER TAS-SPRACHE

Für die Programmiersprache TAS wurde ein Zeichenvorrat gewählt, der es erlaubt, in TAS geschriebene Programme über alle Eingabegeräte in den Rechner einzugeben und auch das Protokoll eines TAS-Quellenprogramms auf allen Ausgabegeräten auszugeben.

Bei der Überschrift, beim Kommentar und bei der Oktadenfolge können weitere Zeichen, soweit sie im Eingabecode vorhanden sind, verwendet werden. Ggf. muß darauf geachtet werden, daß diese Zeichen bei der Ausgabe auch protokolliert werden können.

Eine ganze positive Zahl kann bei einigen Fällen auch durch die ihrem Wert entsprechende Sedezimalzahl ersetzt werden.

$$50 \triangleq '32'$$

Zur Kennzeichnung, daß es sich um eine Sedezimalzahl handelt, wird sie in Apostrophe eingeschlossen.

In einigen Fällen gibt es auch die Möglichkeit, die ganze positive Zahl zur Basis 256 auszudrücken. Hierzu wird ein Zeichen des Zentralcodes verwendet und in jeweils zwei Apostrophe eingeschlossen.

$$''F'' \triangleq 'C5' \triangleq 197$$

Wir sprechen hier von der Oktadendarstellung, da dies intern durch acht Binärstellen dargestellt wird.

7.1. Zeichen

In der Programmiersprache TAS werden 47 Zeichen verwendet, die im Zeichensatz ZS 48 enthalten sind und damit auch in allen Zeichensätzen. Es sind dies

- 10 Ziffern 0 bis 9
- 26 Buchstaben A bis Z
- 12 Sonderzeichen
+ - = . , ' / () * &
und Zwischenraum

Das Steuerzeichen "Wagenrücklauf" (bei Lochkarten entsprechend neue Karte) ist im TAS-Quellenprogramm bedeutungslos und bleibt stets unberücksichtigt (auch in Kommentaren, Überschriften und Oktadenfolgen).

In vielen Fällen kann es nötig sein, den Zwischenraum, der ja durch die Abwesenheit eines Druckzeichens gekennzeichnet ist, durch ein Zeichen darzustellen. Im DIN-Blatt 66 003 wird hier das Zeichen \sqcup oder \bar{b} (von englisch blank) empfohlen.

7.3. Zeichenfolge

Bei den Zeichenfolgen unterscheiden wir zwischen Tetradenfolgen und Oktadenfolgen (Textfolgen siehe unter Konstanten).

Tetradenfolgen sind Sedezimalzahlen, also Zahlen zur Basis 16. Jede Stelle kann daher in vier Binärstellen dargestellt werden und wird auch vom Assembler so übersetzt. Die 16 Möglichkeiten einer Stelle werden durch die Ziffern 0 bis 9 und durch die Buchstaben A bis F dargestellt. Damit Verwechslungen mit Dezimalzahlen ausgeschlossen sind, werden Tetradenfolgen (nicht jede einzelne Stelle) durch Apostrophe eingeschlossen.

7.2. Zahlen

Zahlen können in der üblichen Weise als Dezimalzahlen dargestellt werden. Die dezimale Kommastelle wird dabei stets durch den Dezimalpunkt dargestellt. Die oft übliche Trennung innerhalb der Zahl durch Punkt (z. B. Hunderter- und Tausenderstelle) ist dabei nicht erlaubt. Die Trennung durch Zwischenraum ist nur bei Konstanten erlaubt.

Dezimalzahlen werden als ganze Zahlen mit positiven oder negativen Vorzeichen dargestellt. Das Pluszeichen als Vorzeichen kann entfallen. Des weiteren werden echte und unechte Brüche verwendet.

Oktadenfolgen werden in Doppelapostrophe eingeschlossen. Jede Stelle steht stellvertretend für eine Oktade, das sind 8 Binärstellen. Welchen Binärwert jedem Zeichen der Oktadenfolge zugeordnet wird, kann im allgemeinen dem Zentralcode entnommen werden. Die Zuordnung hängt jedoch auch vom Eingabecode und seiner Umschlüsselung in den Zentralcode ab. Dies kann den Beschreibungen der Eingabevertreter entnommen werden.

7.4. Namen

Namen stehen stellvertretend für eine interne Form. Dies können Adressen von Speicherzellen, Namen von Makros, Namen für Befehle, Namen für Pseudobefehle usw. sein.

Namen beginnen stets mit einem Buchstaben. Es können weitere Buchstaben und Ziffern folgen. Zwischenräume und Sonderzeichen sind nicht erlaubt. Lediglich einige global festgelegte Namen enthalten das Zeichen * (Stern).

Die Länge von Namen ist für den TAS-Assembler auf 31 Zeichen (!) begrenzt. Alle Zeichen eines Namens dienen zur Unterscheidung. Zwei Namen sind nur dann gleich, wenn alle Zeichen gleich sind.

Ausnahmen bilden Montage-, Kontakt- und Zonenamen, von denen nur die ersten 12 Zeichen zur Unterscheidung herangezogen werden. Bei Gebiets- und Lückennamen wirken nur die ersten 6 Zeichen unterscheidend. Von dem Programmnamen werden beim UEBERSETZE-Kommando nur die ersten 12 Zeichen zur Unterscheidung verwendet. Innerhalb des Quellenprogramms gelten alle Zeichen.

7.5. Sonderzeichen als Steuerzeichen

In TAS haben die Sonderzeichen und auch Kombinationen von Sonderzeichen eine spezielle Wirkung. Sie wirken steuernd auf den Übersetzungsvorgang ein.

Nur innerhalb von Oktadenfolgen, Textfolgen, Kommentaren und Überschriften stehen diese Zeichen für sich selbst, wobei einzelne Sonderzeichen oder Kombinationen von Sonderzeichen nicht erlaubt sein können.

Nachstehend sind die Sonderzeichen und ihre Bedeutung erläutert.

+ Plus

Gilt sowohl als Vorzeichen einer positiven Zahl als auch als Verknüpfungszeichen bei der Bildung einer Summe. Als Vorzeichen kann es auch entfallen. (Zahlen ohne Vorzeichen werden als positive Zahlen betrachtet.)

```
+ 186,  
ANF + E,  
ANTON + 3,
```

+ (...) Einschließungszeichen für Makrovariable

Diese Zeichenkombination dient zum Einschluß von Namen für formale Makroparameter (veränderliche Werte innerhalb einer Makrodefinition oder Wiederholungsdefinition).

```
+ (ANFANG),
```

ANFANG = Name des formalen Makroparameters

- Minus

Dieses Zeichen gilt sowohl als Vorzeichen einer negativen Zahl als auch als Verknüpfungszeichen bei der Bildung einer Differenz.

```
- 133,  
ANF - ZAHL,  
NR - 15,
```

-- Einschließungszeichen für Kommentar

Ein Kommentar wird stets durch zwei Minuszeichen eingeleitet und beendet. Innerhalb des Kommentars sind alle im Eingabecode vorhandenen Zeichen erlaubt. Alle Sonderzeichen stehen für sich selbst und haben keine steuernde Wirkung.

Folgende Zeichenkombinationen sind nicht erlaubt:

Minuszeichen am Anfang oder am Ende des Kommentartextes,

zwei aufeinanderfolgende Minuszeichen innerhalb des Kommentartextes.

```
-- SCHLEIFEN - ANFANG --
```

--- Beginn einer Überschrift

Eine Überschrift wird durch drei Minuszeichen eingeleitet. Beendet wird sie, wie beim Kommentar, durch zwei Minuszeichen. Die weiteren Bedingungen sind wie beim Kommentar.

```
--- PROGRAMM - ANFANG ---
```

= Zuordnungszeichen bei Namen

Wird einem Befehl, einer Konstanten oder einem Pseudobefehl ein Name gegeben, so muß er mit dem Gleichheitszeichen abgeschlossen werden. Des weiteren wird das Gleichheitszeichen in der Makrotechnik verwendet, um einem formalen Parameter einen Wert zuzuordnen.

```
ANFANG=...;
B1= ...;
ANF= 30;
ZAHL= 15;
```

• Dezimalpunkt

Wird bei Brüchen an Stelle des sonst üblichen Kommas gesetzt.

```
3.14;
0.185;
-185.4;
```

, Komma

Das Komma zeigt das Ende einer Informationseinheit an und kann entfallen, wenn ein Kommentar oder eine Überschrift folgt.

Innerhalb von Listen (die durch Einschließungszeichen begrenzt sind) trennt es die einzelnen Listenelemente.

```
B ANTON,
AA 15,
C BERTA,
(X1, ANZ, B),
```

' Einschließungszeichen für Tetradenfolgen

Tetradenfolgen müssen, damit sie in jedem Falle von Zahlen und Namen unterschieden werden, in Apostrophe eingeschlossen werden.

```
'A13F',
'100',
'FAB',
```

" Einschließungszeichen für Oktadenfolgen

Oktadenfolgen müssen, damit sie in jedem Fall von anderen Elementen unterschieden werden, in Doppelapostrophe eingeschlossen werden. Das Zeichen Apostroph ist in Oktadenfolgen nicht erlaubt.

```
''TR 440'';
```

'(...)' Einschließungszeichen für Textfolgen

Textfolgen müssen mit der Zeichenkombination Apostroph - runde Klammer beginnen und enden mit der Zeichenkombination runde Klammer - Apostroph.

```
'( ''DM'', ZW, 0, 15)';
```

/ Spezifikationszeichen

Zu Befehlen, Pseudobefehlen und Konstanten kann eine Spezifikation gegeben werden, die eine spezielle Behandlung dieser Informationseinheit steuert.

Die hinter dem Schrägstrich folgenden Zeichen sind Spezifikationen.

```
B ANTON/3VU,
100/MV,
```

(...) Klammern für Listen

In runde Klammern werden Listen oder Literale eingeschlossen.

```
INDEX 10 (X1, ANF, B),
B (65535),
B (1, 2, 4, 8),
```


* Stern

Dieses Zeichen tritt nur bei Namen auf, die von allgemeiner Bedeutung sind und speziell festgelegt wurden. Es darf nur in diesem Fall verwendet werden oder wenn es für sich selbst steht.

```
SFB F*DDV,
```

□ oder b Zwischenraum

Der Zwischenraum ist zur Trennung von Befehlscode und Adressenteil sowie zur Trennung von Links- und Rechtsadressenteil erforderlich. Ansonsten kann er beliebig oft stehen und zu einer übersichtlichen Darstellung beitragen.

```
HXX X1 X2,
XBA ANTON,
```

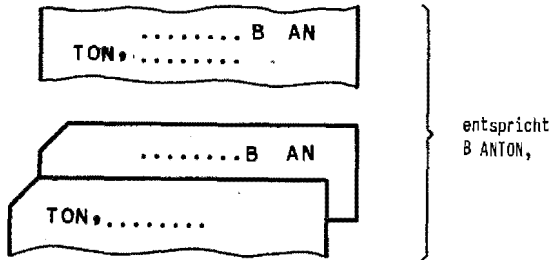
*(...) Stern, runde Klammern

Ein Stern leitet eine Ersetzungsdefinition oder einen Ersetzungsbezug ein. Definition und Bezug sind in runde Klammern eingeschlossen.

```
*(ERS=1, 2, 4, 8),
*(ERS),
S *(ANTON),
```

Kartenende bzw. Zeilenvorschub und Wagenrücklauf

sind für den TAS-Assembler ohne Bedeutung. Sie bilden keine Begrenzung und werden immer ignoriert.



& kommerzielles Und

Dieses Zeichen tritt nur bei Namen auf, die von allgemeiner Bedeutung sind und speziell festgelegt wurden. Es darf nur in diesem Fall verwendet werden oder wenn es für sich selbst steht.

```
B F&RA,
```

TELEFUNKEN
COMPUTER

C

TAS - HANDBUCH

Befehle und Adressierung

INHALT



BEFEHLE UND ADRESSIERUNG 0 - 1

1. OPERATIONSTEIL 1 - 1

- 1. 1. Externcode 1 - 1
- 1. 2. TAS-Interncode 1 - 1

2. GANZADRESSEN 2 - 1

- 2. 1. Adresse eines Ganzwortes 2 - 2
- 2. 2. Adresse eines Halbwortes 2 - 2
- 2. 3. Adressteil als Operand 2 - 2
- 2. 4. Literale 2 - 3

3. LINKS-UND RECHTSADRESSENTEIL 3 - 1

- 3. 1. Indexadresse 3 - 1
- 3. 2. Parameter 3 - 2
- 3. 3. Zweitcodes 3 - 3
- 3. 4. Spezifikationen 3 - 4
- 3. 5. Kernspeicher-Bezug 3 - 5
- 3. 6. Sedezimale 3 - 6

4. ABSOLUTE ANGABEN IM ADRESSENTEIL 4 - 1

- 4. 1. Dezimale Angaben 4 - 1
- 4. 2. Tetradenfolgen 4 - 1
- 4. 3. Oktaden und Textfolgen 4 - 2
- 4. 4. Differenzangaben 4 - 3
- 4. 5. Absolute Summenangaben 4 - 3
- 4. 6. Symbolische Indexadressen 4 - 3

5. SYMBOLISCHE KERNSPEICHERADRESSIERUNG 5 - 1

- 5. 1. Symbolische Adressen 5 - 1
- 5. 2. Relative Adressen 5 - 2
- 5. 3. Summenadressen 5 - 3

6. INDEXZELLENADRESSIERUNG 6 - 1

- 6. 1. Indexadressen 6 - 1
- 6. 2. Pseudobefehl INDEX 6 - 2
- 6. 3. Indexbasis 6 - 5
- 6. 4. Indexadressen bei Quellenprogrammen 6 - 9

7. LITERALE 7 - 1

- 7. 1. Konstanten und Befehle in Literalen 7 - 1
- 7. 2. Literale bei Bringebefehlen 7 - 2
- 7. 3. Literale bei Adressenbefehlen 7 - 3
- 7. 4. Literale bei Sprungbefehlen 7 - 4
- 7. 5. Literale bei Speicherbefehlen 7 - 5
- 7. 6. Unzulässige Anwendung von Literalen 7 - 6

TR 440-Befehle treten als Informationsteil innerhalb einer Informationseinheit auf. Mit TR 440-Befehlen sind hier Maschinen-Befehle gemeint, die vom Assembler, eins zu eins, in die maschineninterne Form übersetzt werden und Anweisungen an die Maschine geben, bestimmte Operationen auszuführen.

Dem Befehl kann ein Benennungsteil vorangestellt sein. Der Benennungsteil gibt dem Befehl einen symbolischen Namen. Unter diesem Namen kann der Befehl im Adressenteil eines anderen Befehls wieder angesprochen werden (siehe Abschnitt "Aufbau" 5.4. Benennung).

Dem Befehl kann sich ein Spezifikationsteil anschließen. Der Spezifikationsteil wird mit einem Schrägstrich eingeleitet; danach folgt eine IE-Spezifikation. Erlaubt sind bei Befehlen nur folgende IE-Spezifikationen:

- K, V, B, D: bestimmen den Ablagebereich
- 0, 1, 2, 3: bestimmen die Typenkennung
- G, U: fordern eine gerade oder ungerade Adresse

Sind keine IE-Spezifikationen hinter einem Befehl angeführt, wird jeder Befehl mit der Typenkennung 2 im Befehlsbereich (B-Bereich) schreibgeschützt abgelegt.

Wird die IE-Spezifikation G angeführt, so erfolgt die Ablage des Befehls auf eine gerade Adresse, bei der IE-Spezifikation U auf eine ungerade Adresse. Ggf. wird vom Assembler der Befehl NULL 0 davorgesetzt, der für die Ausführung ohne Bedeutung ist (siehe Abschnitt "Aufbau", 5.5. IE-Spezifikationen).

Die Informationseinheit wird auf jeden Fall mit einem Abgrenzungsteil abgeschlossen. Der Abgrenzungsteil besteht im einfachsten Fall aus einem Komma; er kann aber auch aus einem Kommentar oder einer Überschrift bestehen (siehe Abschnitt "Aufbau", 5.6. Abgrenzungsteil, 5.7. Kommentar, 5.8. Überschrift).

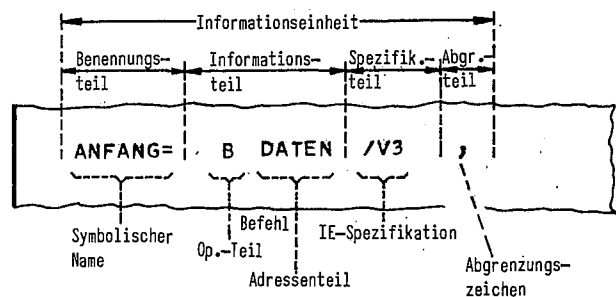


Bild 1.1 Beispiel für die Stellung eines Befehls in einer Informationseinheit

Der eigentliche TR 440-Befehl (der Maschinenbefehl) besteht aus 2 Teilen, aus dem Operationsteil und dem Adressenteil, die in den folgenden Abschnitten beschrieben werden.

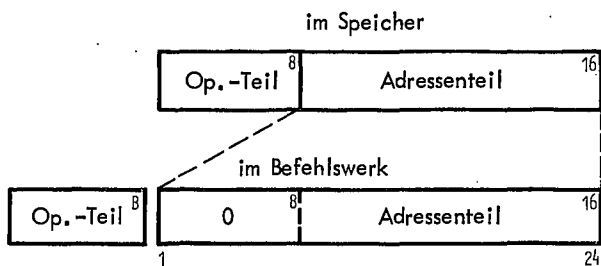


Bild 1.2 Erweiterung des Adressenteils

Intern gesehen ist der TR 440-Befehl eine 24-Bit-Größe; der Befehlscode wird im Operationsteil durch 8 Bits dargestellt. Der nachfolgende Adressenteil ist im Speicher eine 16-Bit-Größe. Beim Transport aus dem Kernspeicher in das Befehlswerk wird der Adressenteil links (binär durch 8 Nullen) auf 24 Bits erweitert.

Für die ursprüngliche 16-Bit-Größe des Adressenteils können 2^{16} Adressen (von 0 bis 65 535) angegeben werden. Durch Modifizierbefehle kann die Adresse in dem auf 24 Bits erweiterten Adressenteil verändert werden. So lassen sich auch größere Adressen als 2^{16} erreichen.

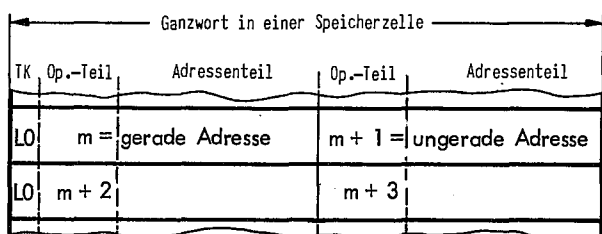


Bild 1.3 Befehlsörter im Speicher

Ein Befehlswort (Ganzwort) in einer Kernspeicherzelle enthält jeweils 2 TR 440-Befehle.

Aus dem Speicher wird stets ein Ganzwort (2 Befehle) ausgelesen. Zunächst wird der Befehl mit der geraden Adresse ausgeführt; danach der mit der ungeraden Adresse. Bei Sprungbefehlen mit erfüllter Sprungbedingung wird das Befehlsfolgeregister neu gesetzt; danach wird ein neues Ganzwort aus dem Speicher geholt, und der angesprungene Befehl mit einer geraden oder ungeraden Adresse kommt zur Ausführung.

1. OPERATIONSTEIL

Für den Operationsteil eines TR 440-Befehls wird ein Befehlscode angegeben. Der Befehlscode kann in externer Form als Externcode oder als TAS-Interncode niedergeschrieben werden.

Der Befehlscode gibt dem Rechner die Anweisung, eine bestimmte Operation auszuführen.

Der Assembler übersetzt den Externcode in eine interne Form. Im Externcode darf anstelle der Ziffer Null auch der Buchstabe O geschrieben werden. Dagegen ist die Ziffer Null anstelle des Buchstabens O ein Fehler.

Beispiel:

TLOG (Tabelle durchsuchen logarithmisch) darf nur so geschrieben werden
aber
SIO (Springe wenn identisch Null) darf auch geschrieben werden
SIO

Daher ist bei der Niederschrift für ein Programm darauf zu achten, daß dieser Unterschied deutlich gemacht wird. Es ist zu empfehlen, den Buchstaben O zu kennzeichnen (z.B. ØTTØ).

Intern gesehen ist der Operationsteil eine 8-Bit-Größe. Durch diese 8 Bits können Größen von 0 bis 255 dargestellt werden, und 256 Befehlscodes lassen sich unterscheiden.

In den Druckschriften "TR 440 Große Befehlsliste" und "TR 440 Befehlslexikon" wird der Interncode in 2 Sedezimalzeichen bei der Beschreibung für jeden Befehl angegeben. Eine Aufstellung von Externcodes mit den entsprechenden Interncodes ist ebenfalls in diesen Schriften zu finden.

1.1. Externcode

Der Externcode besteht aus einem festgelegten Zeichencode und wird vom Assembler in eine interne Anweisung an den Rechner übersetzt, eine bestimmte Operation auszuführen.

Der festgelegte Name eines Externcodes bildet sich aus der mnemotechnischen Abkürzung, welche die Wirkung des betreffenden Befehls in wenigen Worten angibt.

Zum Beispiel:

B2VN Bringe Halbwort mit Vorzeichen negativ

Eine Aufstellung der Externcodes in alphabetischer Reihenfolge sowie eine Gegenüberstellung der Extern- und Interncodes ist in der "TR 440 Große Befehlsliste" und im "TR 440 Befehls-Lexikon" zu finden.

Auf Grund des Externcodes deklariert der Assembler auch den nachfolgenden Adressenteil. Je nach Externcode kann im Adressenteil eine Ganzadresse, eine Rechtsadresse oder eine Links- und Rechtsadresse erwartet werden. An Stelle einer Ganzadresse kann im Adressenteil auch ein Literal stehen.

Die "TR 440 Große Befehlsliste" gibt in der Spalte adr an, welche Angaben für den Adressenteil nach dem jeweiligen Externcode niederschreiben sind.

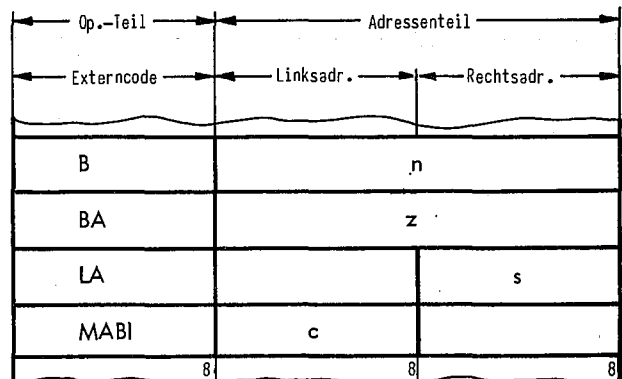


Bild 1.4 Beispiele zeigen Adressenteile in Abhängigkeit von Externcodes.

1.2. TAS-Interncode

Jedem Externcode eines Befehls ist ein Interncode zugeordnet. Der Interncode ist die interne Übersetzung des entsprechenden Externcodes und wird durch 2 Sedezimalzeichen dargestellt. Eine Gegenüberstellung der Externcodes mit den entsprechenden Interncodes ist in den Schriften "TR 440 Große Befehlsliste" und "TR 440 Befehls-Lexikon" zu finden.

Der Interncode kann in der Form des TAS-Interncodes für den Operationsteil eines Befehls niedergeschrieben werden. Die im Abschnitt 1.1. beschriebene Koppelung zwischen Externcode und Adressenteil wird durch die Angabe des TAS-Interncodes aufgehoben. Der Interncode bestimmt zwar die Art der Operation, aber er deklariert nicht den Adressenteil. Der Adressenteil wird in diesem Fall durch den in runden Klammern eingeschlossenen, nachfolgenden Externcode deklariert. Auch wenn der Adressenteil zufällig dem Interncode entspräche, muß bei der Darstellung eines TAS-Interncodes der Adressenteil durch den in runden Klammern eingeschlossenen Externcode bestimmt werden. Eine weitere Bedeutung für die Operation hat der Externcode in diesen Fällen nicht.

Die Niederschrift eines TAS-Interncodes erfolgt in folgender Form:

I 'i'(e) adr, i: Interncode
 e: Externcode
 adr: Adressenteil

Der Buchstabe I deklariert den folgenden Teil als TAS-Interncode. Anschließend erfolgt die Angabe des Interncodes in 2 Sedezimalzeichen. Die Sedezimalzeichen müssen von Einschließungszeichen (Apostroph) eingeschlossen sein. Darauf erfolgt in runden Klammern eingeschlossen der Externcode, welcher den nachfolgenden Adressenteil bestimmt. Die Informationseinheit wird wie üblich mit einem Abgrenzungsteil abgeschlossen.

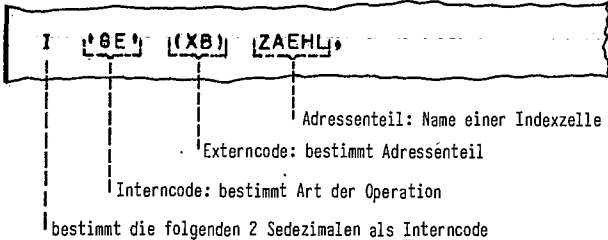


Bild 1.5 Beispiel für die Darstellung eines TAS-Interncodes

Das im Bild 1.5 gezeigte Beispiel für die Darstellung eines TAS-Interncodes wird hier erläutert.

Der Buchstabe I gibt an, daß die nachfolgenden Sedezimalzeichen '8E' einen Interncode darstellen. Der Interncode entspricht dem Externcode des Befehls BA (Bringe Adressenteil). Ursprünglich ist für diesen Befehl im Ganzadressenteil die Angabe des Wertes z vorgesehen. Der Adressenteil wird aber in diesem Fall durch den in runden Klammern eingeschlossenen Externcode XB als Adresse einer Indezelle deklariert. Die Adresse dieser Indezelle ist hier mit ZAEHL angegeben.

Bei diesem Beispiel wird durch die Darstellung des TAS-Interncodes '8E' die Adresse der Indezelle ZAEHL in das Register A gebracht (und nicht der Inhalt der Indezelle in das Register B).

Die Anwendung des TAS-Interncodes eignet sich auch zur Darstellung von Konstanten, die eines der zahlreichen Befehlsformate haben. Diese Möglichkeit kann z.B. genutzt werden, um Versorgungsblöcke darzustellen, wo Befehlscode und Adressenteil einander nicht entsprechen.

Siehe hierzu Abschnitt "Konstanten" 1.9. Konstanten im Befehlsformat.

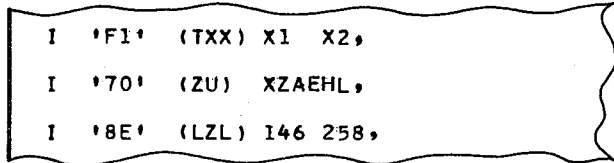


Bild 1.6 Beispiele für die Niederschrift bei der Darstellung von TAS-Interncodes

2. GANZADRESSE

Die Ganzadresse eines Befehls umfaßt den ganzen 16-Bit-Adressenteil. (Der Begriff Ganzadresse ist nicht gleichbedeutend mit den Begriffen Ganzwortadresse oder Ganzwort!)

In Ganzadressen können die Adressen von Ganzwörtern (n), die Adressen von Halbwörtern (m) und Operanden (z) dargestellt werden.

Intern gesehen können in der 16-Bit-Größe einer Ganzadresse Werte von 0 bis 65 535 angeführt werden. Diese Werte lassen sich durch Modifizierung des in der Abrufphase auf 24 Bits verlängerten Adressenteils verändern (erhöhen). Siehe dazu Bild 1.2.

Externcode	Ganzadresse	Art der Adresse
B	n	Adresse eines Ganzwortes
B2	m	Adresse eines Halbwortes
B3	m	Adresse eines Halbwortes
AA	z	Zahl (oder Operand)
Op.-Teil	8	Adressenteil 16

Bild 2. 1 Beispiele verschiedener Ganzadressen

Für eine Ganzadresse kann ein absoluter Bezug oder ein Kernspeicherbezug niedergeschrieben werden. Anstelle einer Ganzadresse darf auch ein Literal stehen. Absolute Bezüge werden vom Assembler in Zahlen zwischen 0 und 65 535 übersetzt und geben so absolute Adressen an. Diese Adressen werden auch bei späterer Montage nicht mehr verändert. Absolute Bezüge lassen sich sinnvoll bei Befehlen verwenden, in deren Adressenteil z angegeben ist. Sollen absolute Bezüge zur Adressierung von Ganzwörtern (n) oder Halbwörtern (m) dienen, ist diese Verwendung nur sinnvoll, wenn die absoluten Speicheradressen bei der Niederschrift für ein Programm erkennbar sind (siehe Abschnitt 4. Absolute Speicheradressierung).

Indexnamen bezeichnen bestimmte Indexzellen und sind stets absolute Bezüge, die vom Assembler in Zahlen übersetzt, in der Ganzadresse stehen.

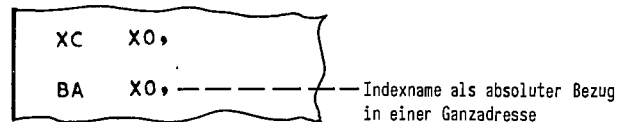


Bild 2.2 Beispiel für die Darstellung eines Indexnamens in einer Ganzadresse

Im Beispiel Bild 2.2, wird ein Indexname als absoluter Bezug in einer Ganzadresse verwendet. Dieser Name muß vorher definiert sein. In diesem Beispiel setzt der Assembler auf Grund des Befehls XC für den Indexnamen X0 die Adresse einer Indexzelle ein. Bei dem Befehl BA übersetzt der Assembler den Namen der Indexzelle in die gleiche Adresse.

Wird eine Indexzelle mit Hilfe des Pseudobefehls INDEX deklariert, kann dieser Name beliebig als absoluter Bezug in einer Ganzadresse verwendet werden. Er braucht beim Beispiel in Bild 2.2 nicht vor dem Befehl BA definiert zu sein, sondern kann an beliebiger Stelle des Programms durch den Befehl INDEX deklariert werden (siehe Abschnitt 6. Indexadressierung).

Kernspeicherbezüge dienen zur Adressierung von Befehlen, Konstanten oder Speicherbereichen. Wird ein Kernspeicherbezug dargestellt, prüft der Assembler, ob dieser Bezug den TAS-Konventionen entspricht und somit zulässig ist. (Nicht zulässig sind z.B. Abspeicherungen in schreibgeschützte Bereiche.)

Ein Kernspeicherbezug kann bei der Übersetzung noch nicht in eine absolute Adresse, sondern nur in eine zum Quellenprogramm relative Adresse umgewandelt werden. Die Übersetzung in eine absolute Adresse erfolgt erst bei der Montage (siehe Abschnitt 5. Symbolische Kernspeicheradressierung).

Die Darstellung von Literalen, die anstelle einer Ganzadresse stehen, werden im Abschnitt 2.4. beschrieben.

2.1. Adresse eines Ganzwortes

Die Adresse eines Ganzwortes wird in einer Ganzadresse dargestellt. In den Druckschriften "TR 440 Große Befehlsliste" und "TR 440 Befehls-Lexikon" ist in den Spalten Adressenteil (adr) bei den Befehlen, die eine Ganzwortadresse haben, ein n eingetragen. Die Adressierung des Ganzwortes erfolgt in gleicher Form wie sie im Abschnitt 2. Ganzadresse beschrieben ist (siehe auch Abschnitte 4. und 5.).

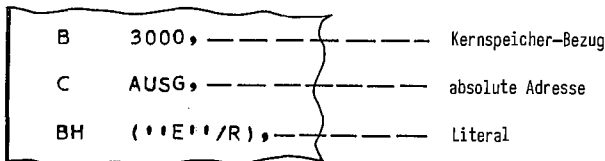


Bild 2.3 Beispiele von Ganzwortadressen

Intern gesehen können für die Adresse eines Ganzwortes im 16-Bit-Adressenteil eines TR 440-Befehls Werte geradzahlig von 0 bis 65 534 angegeben werden.

Bei ungeraden Adressen ist bei Ansteuerung des Speichers die Wirkung so, als würden die Adressen um 1 vermindert. Durch Modifizierung kann die Ganzadresse verändert werden.

Geradzahlige Adressen bezeichnen das linke Halbwort und ungeradzahlige das rechte Halbwort in der Speicherzelle.

Die Darstellung der Halbwortadressen erfolgt in der gleichen Form wie die Darstellung von Ganzwortadressen. Für die Niederschrift gelten die gleichen Angaben, die im Abschnitt 2 allgemein für alle Ganzadressen aufgeführt sind.

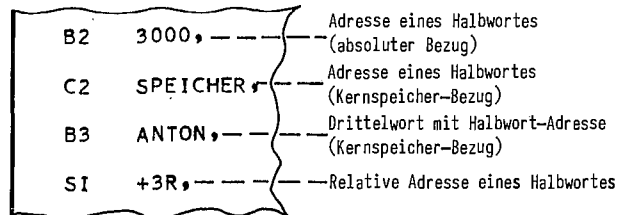


Bild 2.5 Beispiele für die Niederschrift von Halbwort-Adressen

Intern gesehen können im 16-Bit-Adressenteil einer Ganzadresse 65 535 Halb- oder Drittelwörter adressiert werden. Durch Modifizierung können die angegebenen Werte in der Ganzadresse verändert werden.

2.2. Adresse eines Halbwortes

Die Adresse eines Halbwortes wird (ebenso wie die Adresse eines Ganzwortes) in einer Ganzadresse dargestellt. In den Schriften "TR 440 Große Befehlsliste" und "TR 440 Befehls-Lexikon" ist bei den Befehlen, die im Adressenteil die Adresse eines Halbwortes fordern, in der Spalte Adressenteil (adr) ein m aufgeführt. Mit der Adresse eines Halbwortes wird auch ein Drittelwort angesprochen. (Ein Drittelwort umfaßt den rechten 16-Bit-Teil eines Halbwortes; ein Ganzwort besteht also nicht aus 3 Drittelwörtern.)

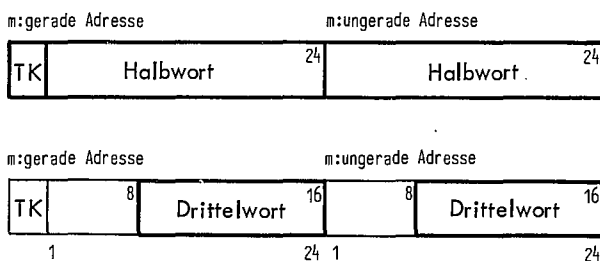


Bild 2.4 Halbwörter und Drittelwörter mit geraden und ungeraden Adressen im Speicher

2.3. Adressenteil als Operand

Bei einer Anzahl von Befehlen wird in einer Ganzadresse ein Operand dargestellt. Im "TR 440 Befehls-Lexikon" und in der "TR 440 Große Befehlsliste" ist bei diesen Befehlen in den Spalten Adressenteil (adr) ein z angeführt; z steht hier für Zahl oder Operand. Es wird hier der Operand selbst niedergeschrieben und nicht die Adresse des Operanden. Der Operand darf aber auch eine Adresse sein. Der Operand kann als Zahl mit dem absoluten Wert zwischen 0... 65 535 dargestellt werden. (Die Befehle AA und SBA machen eine Ausnahme. Der Wert, der hier für den Operanden eingesetzt werden darf, ist den Befehlsbeschreibungen im "TR 440 Befehls-Lexikon" zu entnehmen.) Wie im Abschnitt 2. beschrieben, kann auch für einen Operanden im Adressenteil ein Kernspeicherbezug aufgeführt werden. In diesem Fall wird die Adresse, (in welche der Kernspeicherbezug bei der Montage übersetzt wird) als Operand benutzt. Es gilt auch hier, daß der für den Kernspeicherbezug übersetzte Wert maximal 65 535 (dezimal 'FFFF') nicht überschreiten darf (siehe Abschnitt 3.6.).

Außerdem gibt es die Möglichkeit, bei Befehlen mit Ganzadresse, einen Operanden mit Hilfe eines Literals darzustellen (siehe Abschnitt 2.4.).

A=	AA	200/K,
	HBA	35000,
	VBA	A+2,
	BA	'10'+B-A,
B=	MA	100/K,

Bild 2.6 Beispiele für Operanden im Adressenteil

2.4. Literale

Bei Befehlen, für die im 16-Bit-Adressenteil die Angabe einer Ganzadresse (n, m oder z) erwartet wird, kann statt einer Ganzadresse ein Literal angegeben werden. Mit der Darstellung eines Literals ist die Möglichkeit gegeben, anstelle der Adresse eines Operanden, den Operanden selbst darzustellen.

Literale können eine Informationseinheit oder eine Folge von Informationseinheiten darstellen. Die Form des Literals stellt einen Bezug auf die erste Informationseinheit des Literals dar. Die Adresse dieser ersten Informationseinheit wird bei der Übersetzung in den Adressenteil des bezugnehmenden Befehls eingesetzt.

Für die Niederschrift eines Literals gilt folgende Form:

e (i),
 e: Externcode
 i: Informationseinheit(en)

Nach Angabe des Externcodes folgt hinter einer runden Klammer die Darstellung der ersten Informationseinheit des Literals. Jede weitere Informationseinheit innerhalb des Literals wird von der nachfolgenden durch ein Komma abgegrenzt. Die letzte Informationseinheit des Literals wird durch die abschließende runde Klammer begrenzt. Literale sparen in bestimmten Fällen Schreibarbeit und dienen zur Erhöhung der Übersicht.

Die Verwendung von Literalen wird im Abschnitt 7. eingehend beschrieben.

3. LINKS- UND RECHTSADRESSENTEIL

C

Bei einer Anzahl von Befehlen wird für den Adressenteil die Angabe von 2 Werten gefordert. Der Adressenteil teilt sich in diesen Fällen in einen Links- und einen Rechtsadressenteil. Bei einer weiteren Anzahl von Befehlen wird auch nur eine Angabe im Rechtsadressenteil erwartet.

In den Schriften "TR 440 Große Befehlsliste" und "TR 440 Befehls-Lexikon" ist jeweils angeführt, welche Angaben in den Adressenteilen gefordert werden.

Folgende Zeichen werden in den Adressenteilen aufgeführt:

- i: Indexadresse
 - p: Parameter
 - s: Spezifikation
(s evtl. unterteilt in s_1, s_2, \dots)
 - c: Zweitcode (Code für den Zweitbefehl)
- Indizes: L für links
R für rechts

Externcode	Adressenteil		Art der Adressenteile	
MFU		i		Indexbezug
KDFR		P		Parameter
RT		s		Spezifikation
LR		$s_1 s_2$		2 Spezifikationen
ZX	p	i	Parameter,	Indexbezug
TXR	s	i	Spezifikation,	Indexbezug
E	c	i	Zweitcode,	Indexbezug
VXX	i_L	i_R	Indexbezüge in beiden Adressenteilen	
SHB	s	p	Spezifikation,	Parameter
MAB	c	p	Zweitcode,	Parameter
SEGG	p_L	p_R	Parameter in beiden Adressenteilen	
SL	p	s	Parameter,	Spezifikation
LZL	s_L	s_R	Spezifikationen in beiden Adressenteilen	
RLR	c	s	Zweitcode,	Spezifikation
	8	8	8	
Op.-Teil	Linksadr.	Rechtsadr.		
1	8	24		

Für jedes dieser Zeichen muß ein Wert niedergeschrieben sein; auch wenn die Angabe für die Art bestimmter Operationen keine Bedeutung haben sollte, erwartet der Assembler diese Angabe. Ggf. ist der Wert Null anzugeben.

Der Assembler übersetzt die Angaben für die Adressenteile und prüft, ob Parameter innerhalb des zulässigen Bereichs liegen, ob Zweitcodes, Spezifikationen und Kernspeicherbezüge den Konventionen entsprechen und zulässig sind.

Werden die Angaben für den Adressenteil in interner Form, z.B. mit 2 Sedizimalzeichen angegeben, entfällt die Prüfung auf Zulässigkeit (siehe Abschnitt 3.6. Sedezimale).

Intern gesehen sind beide Adressenteile 8-Bit-Größen. In jedem Adressenteil lassen sich Werte von 0 bis ± 127 oder von 0 bis 255 darstellen. Die jeweils zulässigen Wertangaben sind den Befehlsbeschreibungen zu entnehmen.

Bild 3.1 Beispiele für Links- und Rechtsadressen

IAS - Handbuch

Juli 69

3.1. Indexadresse

Für eine Anzahl von TR 440-Befehlen wird im Links- oder Rechtsadressenteil, oder in beiden Adressenteilen gleichzeitig, die Darstellung einer Indexadresse erwartet. Eine Indexadresse wird mit einem Indexbezug bezeichnet. Dieser Indexbezug stellt einen absoluten Bezug dar, der vom Assembler in eine absolute Adresse einer Indexzelle zwischen 0 und 255 relativ zum Anfang der Indexzone übersetzt wird. Die insgesamt 256 Indexzellen liegen hintereinander in einem bestimmten Teil des Kernspeichers, der sogenannten Indexzone (siehe Abschnitt 6. Indexadressierung).

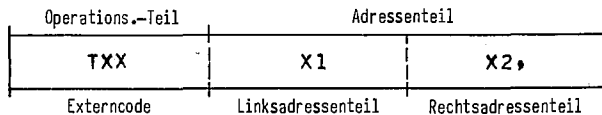


Bild 3.2 Indexadressen in den Adressenteilen

3.2. Parameter

Bei einer Anzahl von TR 440-Befehlen wird die Darstellung von Parametern in den Adressenteilen erwartet. In den Schriften "TR 440 Große Befehlsliste" und "TR 440 Befehls-Lexikon" ist bei diesen Befehlen in den Spalten Adressenteil (adr) p eingetragen. Ein Parameter kann je nach Befehl im Links- oder Rechtsadressenteil oder in beiden Adressenteilen gleichzeitig vorgesehen sein.

Externcode	Adressenteil		Bereich für p
ZX	p	i	p: $\pm 0 \dots \pm 127$
SHB	s	p	p: Schrittschritte 0...255
SEGG	P _L	P _R	p_L : Sprungweite $\pm 0 \dots \pm 127$ p_R : Vergl.-Expon. $\begin{cases} 0 \dots 127 \text{ positiv} \\ 00 \dots N127 \text{ negativ} \end{cases}$
	8	8	8
	Op.-Teil	Linksadr.	Rechtsadr.

Bild 3.2 Beispiele von Parametern in verschiedenen Adressenteilen

Bei den Befehlsbeschreibungen ist jedesmal angegeben, in welchem Bereich die Wertangaben für den Parameter liegen dürfen.

Parameter werden zum Beispiel benötigt, um bei Sprungbefehlen mit relativen Sprung- (Ziel-) Adressen die Sprungweite anzugeben. Schifftbefehle benötigen in ihrem Adressenteil einen Parameter, um die Schifftschritte darzustellen. Als Addend wird der Parameter bei Ersetzbefehlen verwendet, um den Inhalt der Register B oder U zu erhöhen. Beim Modifizierbefehl MH wird mit Hilfe eines Parameters der Inhalt einer Indexzelle erhöht und als Modifikator bereitgestellt. Bei einigen anderen Befehlen wird ebenso durch einen Parameter der Inhalt von Indexzellen verändert. Bei den Konvertierungsbefehlen KDFR und KFLD dienen Parameter dazu, die gewünschten Dezimalstellen anzugeben, die konvertiert werden sollen.

Die Wertangaben für Parameter sind absolute Wertangaben. Sie werden bereits bei der Übersetzung in Zahlen zwischen 0 und 255 (oder $-127 \dots +127$) umgewandelt. Diese Zahlen geben z.B. bei Sprungbefehlen die Adressen des anzuspringenden Befehls relativ zum bezugnehmenden Befehl an oder sie stellen eine absolute Zahl für Operationen bei anderen Befehlen dar.

Für die Niederschrift von Parametern gilt folgendes:

Parameter können als Zahlen (dezimal) mit Vorzeichen dargestellt werden. Bei positiven Werten kann das Vorzeichen entfallen. Für den Befehl SEGG gilt eine Ausnahme. Hier wird im Adressenteil die Angabe von 2 Parametern gefordert (P_L P_R). Für P_R werden positive Parameter ohne Vorzeichen, bei negativen Werten werden die Parameter mit einem vorangestellten N angegeben (positiv: 0...127; negativ: N0...N127). Diese Ausnahme ist erforderlich, damit die folgende beschriebene Summen- oder Differenzbildung der beiden Parameter unterbleibt. Der Assembler übersetzt die dezimale Angabe in die interne Form und prüft, ob der Wert jeweils im zulässigen Bereich liegt. Ein Parameter kann auch als Tetraden- oder Oktadenfolge oder als Textfolge angegeben werden. Wird er in interner Form dargestellt, so unterbleibt die Prüfung auf Zulässigkeit

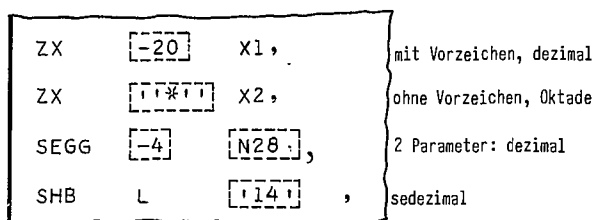


Bild 3.3 Darstellung von Parametern in dezimaler und sedezimaler Form und als Oktadenfolge

Weitere Möglichkeiten für die Niederschrift von Parametern werden durch Summen- oder Differenzangaben geboten.

Wird ein Wert mit einem anderen Wert durch ein Pluszeichen verbunden, so ermittelt der Assembler die Summe und setzt diese bei der Übersetzung in den Adressenteil des Befehls ein.

Werden zwei Werte durch ein Minuszeichen verbunden, so ermittelt der Assembler die Differenz und setzt diese als absoluten Wert im Adressenteil ein. Auch für die vom Assembler ermittelten Werte darf der zulässige Bereich des jeweiligen Parameters nicht überschritten werden (siehe Abschnitt 4.2., Differenzadressen und 4.3. absolute Summenadressen).

Niederschrift:

ZX	[54]-[4F]	X1,
ST	[*]-[4F]	1A,
HXP	[2+4F]	X2,
SHB	R	[14]+4,

nach Assemblierung:

ZX	[05]	X1,
ST	[29]	A1,
HXP	[51]	X2,
SHB	R	[18],

Bild 3.4 Beispiele für die Übersetzung von Parametern bei Angaben zweier Werte durch Summen- und Differenzbildung

In bestimmten Fällen ist es von Vorteil, die Darstellung von Parametern in Form von Kernspeicherbezügen vorzunehmen. Treten in Linksadressenteilen Kernspeicherbezüge auf, werden sie vom Assembler als Differenz zwischen Zieladresse und Adresse des bezugnehmenden Befehls übersetzt. Diese Differenz muß jedoch in einem für den Parameter zulässigen Bereich liegen (siehe Abschnitt 3.5. und 5.).

A=	M	X1,
	STN	E 1H,
E=	SZX	A X1,

Bild 3.5 Beispiele für Parameter in Linksadressenteilen die in Form von Kernspeicher-Bezügen dargestellt sind

3.3. Zweitcodes

Ist für den Operationsteil eines TR 440-Befehls der Externcode eines Ersetzbefehls angegeben, wird im Linksadressenteil die Darstellung eines Zweitcodes erwartet. Dieser Zweitcode gibt den Befehl an, der anstelle des Ersetzbefehls zur Ausführung kommt.

Da im Normalfall alle Befehle schreibgeschützt im Kernspeicher abgelegt werden, können Ersetzbefehle benutzt werden, um den Adressenteil des Befehls vom Programm selbst zu ändern. Im Ablauf des Programms wirkt sich dies so aus, als ob der Ersetzbefehl durch den von ihm erzeugten Befehl ersetzt wird und an der gleichen Stelle des Programms steht.

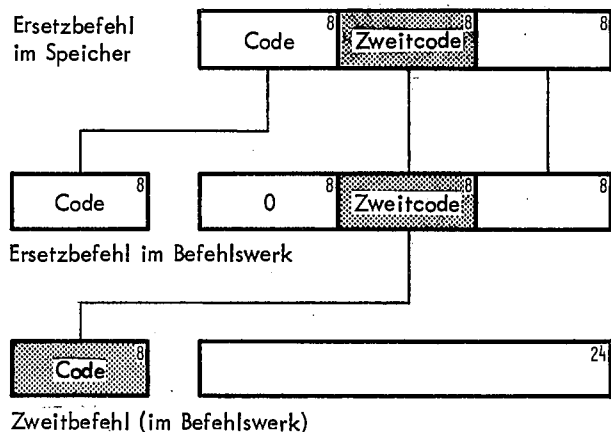


Bild 3.6 Prinzip der Doppelcodebefehle (Ersetzbefehle)

Der Adressenteil des zur Ausführung kommenden Zweitbefehls ist abhängig vom Befehlscode des Erstbefehls. Der Assembler erwartet nach der Darstellung des Zweitcodes im Linksadressenteil für den Rechtsadressenteil entweder die Adresse einer Indexzelle (i) die Darstellung eines Parameters (p) oder eine Spezifikation (s).

Externcode	Adressenteil		Art des Adressenteils
E	c	i	Zweitcode, Indexadr.
MAB	c	p	Zweitcode, Parameter
R	c	s	Zweitcode, Spezifikation
Op.-Teil ⁸	Linksadr. ⁸	Rechtsadr. ⁸	

Bild 3.7 Beispiele von Angaben in den Adressenteilen von Ersetzbefehlen

In den Schriften "TR 440 Befehls-Lexikon" und "TR 440 Große Befehlsliste" ist bei den Ersetzbefehlen (E, EZ, ENZ, EMB, MAB, MU, EMU, RLR und R) in den Spalten Adressenteil (adr) für den Linksadressenteil c eingetragen. Der Zweitcode ist durch jeden TR 440-Befehl darstellbar. Eine Ausnahme macht nur der Befehl R. Welcher Zweitcode beim Befehl R zugelassen ist, wird in den aufgeführten Schriften bei jedem Befehl angezeigt. Eine Zusammenstellung dieser Befehle ist der Beschreibung des Befehls R angefügt (siehe Abschnitt Adressenrechnung Ersetzbefehle).

Externcode	Zweitcode	Angaben für den Adressenteil	
EMB	B	X1,	Befehl B Indexzelle X1,
EZ	C	X2,	Befehl C Indexzelle X2,
E	'42'	X3,	Befehl A Indexzelle X3, in Sedezimalzeichen
MAB	B	4,	Befehl B Parameter =4,
MU	S	3,	Befehl S Parameter =3,
R	A	H,	Befehl A Spezifikationen H = Register,
Op.-Teil	Linksadr.	Rechtsadr.	

Bild 3.8 Beispiele für die Niederschrift von Zweitcodes

Bei der Niederschrift kann der Zweitcode auch in der internen Form mit 2 Sedezimalzeichen dargestellt werden. Die für den TAS-Intercode geforderte Kennzeichnung durch den Buchstaben I ist hier nicht erlaubt.

3.4. Spezifikationen

Für bestimmte Befehle werden im Links- oder Rechtsadressenteil oder gleichzeitig in beiden Adressenteilen Spezifikationen erwartet. Die in diesen Adressenteilen

dargestellten Spezifikationen variieren auf spezielle Art die Wirkung der betreffenden Operation; sie dienen aber nicht der Adressierung.

Externcode	Adressenteil		Art der Spezifikationen
RT		s ₁ s ₂	s ₁ , s ₂ : A, Q, D oder H; s ₁ ≠ s ₂
LR		s ₁ s ₂	s ₁ : 0, 1, 2 oder 3 (Typenennung) s ₂ : A, Q, D oder H (Register)
TXR	s ₁ s ₂	i	s ₁ : A, Q, D und H (Register) s ₂ : leer = +; N = - (± (i))
LZL	s _L	s _R	s: Merklichter 0, 1, 2... und 8 0 bedeutet kein Merkllicht s _L : = L, s _R : = 0
Op.-Teil ⁸	Linksadr. ⁸	Rechtsadr. ⁸	

Bild 3.9 Beispiele für Spezifikationen in den verschiedenen Adressenteilen

In den Schriften "TR 440 Befehls-Lexikon" und "TR 440 Große Befehlsliste" ist für diese Befehle in den Spalten Adressenteil (adr) s eingetragen. Die für s einzusetzenden festgelegten Buchstaben und Ziffern werden dort erläutert. Ist für s z.B. A, Q, D und H eingetragen, so können mehrere dieser Angaben für s eingesetzt werden.

Ist für s A, Q, D oder H eingetragen, so darf nur eine dieser Angaben für s verwendet werden.

Wird in s₁, s₂ usw. unterteilt, so können für jedes s_x Angaben eingesetzt werden.

Die interne Darstellung dieser Spezifikation ist für die betreffenden Befehle ebenfalls aus diesen Schriften zu ersehen.

Niederschrift:

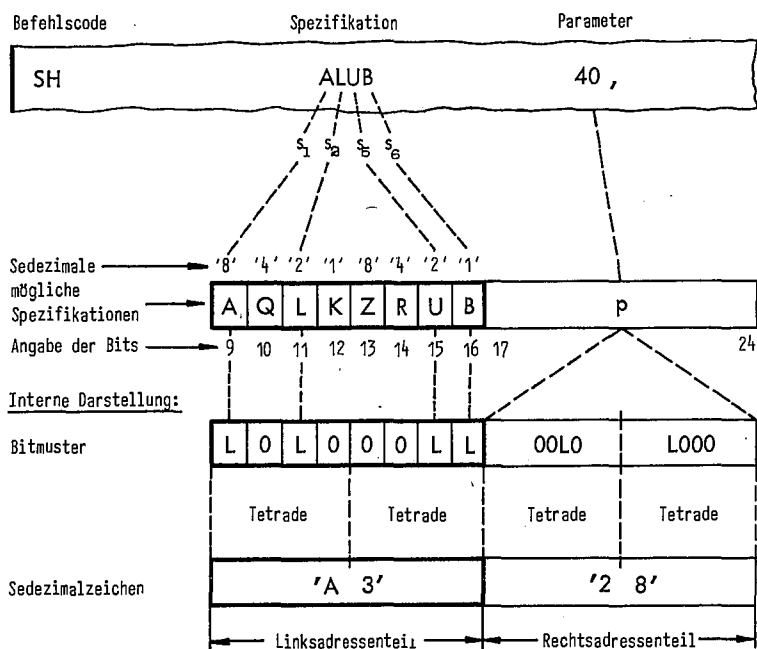


Bild 3.10 Beispiel für Spezifikationen in verschiedenen Darstellungs-Formen beim Befehl SH (zu diesem Beispiel siehe auch die Schrift "TR 440 Große Befehlsliste" Seite Internspezifikationen)

Der Assembler übersetzt die angegebenen Spezifikationen in die maschineninterne Form und prüft die Zulässigkeit der Angaben. Erfolgt die Darstellung bei der Niederschrift in einer internen Form, so unterbleibt die Zulässigkeitsprüfung.

3.5. Kernspeicherbezug

Ein Kernspeicherbezug steht als Name symbolisch für die Adresse einer Kernspeicherzelle. Die absolute Adresse dieser Kernspeicherzelle wird vom Assembler erst bei der Montage ermittelt.

Treten Kernspeicherbezüge in Linksadressenteilen von Befehlen auf, so ersetzt der Assembler sie in relative Adressen.

Diese Möglichkeit kann bei relativen Sprungbefehlen genutzt werden, in deren Linksadressenteil Parameter für die Sprungweite anzugeben sind. Der Assembler bildet die Differenz zwischen der Adresse des Befehls, in dessen

Linksadressenteil der symbolische Name auftritt und der Zieladresse, das ist der Befehl, dem der gleiche symbolische Name vorangestellt ist. Diese Differenz kann nur gebildet werden, wenn beide Adressen in der gleichen Adressenzone stehen. Die Differenz steht nach der Übersetzung im Linksadressenteil des Befehls als Parameter für die Sprungweite. Diese Sprungweite ist relativ zum Befehl selbst.

Dieser so ermittelte Differenzwert darf den jeweils zulässigen Bereich des betreffenden Parameters nicht überschreiten und wird vom Assembler in maschineninterner Form in den 8 Bits des Linksadressenteils dargestellt.

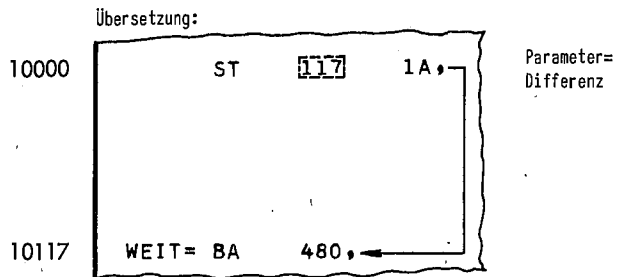
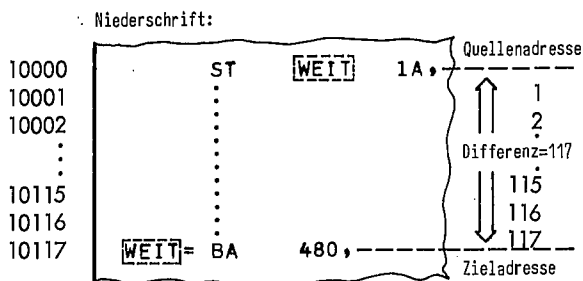


Bild 3.11 Beispiel für die Differenzbildung

Diese Möglichkeit ist von Vorteil, wenn bei der Niederschrift für ein Programm der Abstand zwischen den Befehlen nicht bekannt ist. Bei Änderungen im Programm, wobei der Abstand zwischen den Befehlen verändert wird, muß die Angabe der Sprungweite in diesem Fall nicht korrigiert werden, da der Assembler die jeweilige Differenz selbst ermittelt.

Bei dem in Bild 3.11 dargestellten Beispiel steht im Linksadressenteil des Befehls ST als Kernspeicherbezug der Name WEIT. Dieser Name steht für einen Parameter, der die Sprungweite angeben soll. Der gleiche Name ist dem Befehl BA vorangestellt. Der Assembler ermittelt die Differenz zwischen den Befehlen und setzt diese in den Linksadressenteil des Befehls ST ein.

3.6. Sedezimale

Sedezimalzeichen stellen einen maschineninternen Code dar. Ein Sedezimalzeichen steht jeweils für eine Tetrade, eine Kombination von 4 Bits.

Mit Sedezimalzeichen von 0 - F lassen sich, wie die Tabelle zeigt, dezimale Werte von 0 bis 15 darstellen. Dies entspricht der Bitkombination von 0000 bis LLLL.

Werden für Links- und Rechtsadressenteile 2 Sedezimalzeichen angeführt, entfallen bei Übersetzung alle Prüfungen auf Zulässigkeit.

Intern gesehen beansprucht ein TR 440-Befehl den Raum von 6 Tetraden. Der Links- und Rechtsadressenteil kann durch je zwei Sedezimalzeichen dargestellt werden.

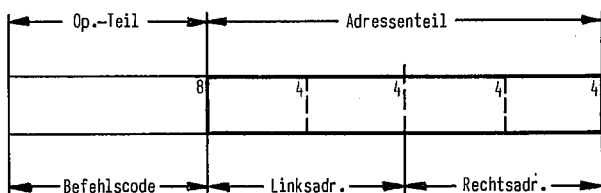


Bild 3.12 Darstellung einer Tetradeinteilung für den Adressenteil eines Befehls im Speicher

Sedezimal	Dezimal	Tetrade (4 Bit-Gruppe)
'0'	0	0 0 0 0
'1'	1	0 0 0 L
'2'	2	0 0 L 0
'3'	3	0 0 L L
'4'	4	0 L 0 0
'5'	5	0 L 0 L
'6'	6	0 L L 0
'7'	7	0 L L L
'8'	8	L 0 0 0
'9'	9	L 0 0 L
'A'	10	L 0 L 0
'B'	11	L 0 L L
'C'	12	L L 0 0
'D'	13	L L 0 L
'E'	14	L L L 0
'F'	15	L L L L

Bild 3.13 Sedezimalziffern und entsprechende Dezimalzahlen und Bitmuster

Für die Niederschrift von Sedezimalzeichen gilt folgende Form:

's' s: Sedezimalzeichen;
 0, 1, 2, 3, 4, 5, 6, 7, 8, 9,
 A, B, C, D, E, F,

		Sedezimale für:
HXP	'0C' X1,-----	- einen Parameter;
ST	'7D' '06',-----	- einen Parameter und Spezifikationen;
SEGG	'5B' '3A',-----	- zwei Parameter;
MU	'36' '01',-----	- einen Zweitcode, einen Parameter;
SH	'F2' 12,-----	- Spezifikationen und einen Parameter.

Bild 3.14 Beispiele für die Schreibweise von Spezifikationen in Links- und Rechtsadressenteilen

Bei den Beispielen in Bild 3.14 entsprechen die Sedezimalzeichen beim Befehl ST den Spezifikationen IA, (I für s_1 : Typenkennung = 1, A für s_2 : Register A); beim Befehl SH den Spezifikationen A, Q, L, K, U (AQ für s_1 , L für s_2 , K für s_3 , U für s_5).

C

4. ABSOLUTE ANGABEN IM ADRESSENTEIL

C

Werden im Adressenteil eines Befehls absolute Angaben gemacht, so müssen sie vom Assembler bereits bei der Übersetzung (dezimal gesehen) in Zahlen zwischen 0 und 65 535 übersetzt werden können. Diese Werte werden auch bei einer späteren Montage nicht mehr verändert.

Absolute Angaben lassen sich im allgemeinen nur bei Befehlen verwenden, in deren Adressenteil ein Wert für das Zeichen z (Zahl oder Operand) angegeben wird.

Diese Angaben können extern in folgenden Formen dargestellt werden:

Dezimale Angaben,
Tetradenfolgen,
Oktaden- oder Textfolgen,
Differenzangaben,
absolute Summenangaben,
symbolische Indexadressen.

In den folgenden Abschnitten werden die verschiedenen Formen von absoluten Angaben beschrieben.

Wenn hier von "absoluten Adressen" die Rede ist, so sind operatorrelative Adressen gemeint. Die Zuordnung von operatorrelativen Adressen zu absoluten Adressen erfolgt automatisch mit Hilfe des Betriebssystems durch die Hardware. Diese Tatsache ist für den Programmierer jedoch unbedeutend.

4.1. Dezimale Angaben

Dezimalzahlen lassen sich im Bereich von 0 bis 65 535 in einer Ganzadresse darstellen. Die Zahlen werden als Ziffernfolge hintereinander, ohne Zwischenräume, niedergeschrieben. Zwischenräume trennen die Ganzadresse und führen zu Fehlermeldungen. Der Assembler übersetzt die dezimalen Angaben in die entsprechenden Bitmuster (Bild 4.2) und setzt diese rechtsbündig in den Ganzadressenteil ein.

BA	32768,
VBA	16384,
ETA	8192,
AA	48,

Bild 4.1 Beispiele für dezimale Angaben in Ganzadressen

4.2. Tetradenfolgen

Absolute Angaben in Ganzadressen lassen sich als Tetradenfolgen darstellen. Sie werden in Form von Sedezimalziffern zur Basis 16 dargestellt. Jede der 16 möglichen Ziffern steht für eine Kombination von 4 Bits (eine Tetrade).

Sedezimal	Dezimal	Tetrade (4Bit-Gruppe)
'0'	0	0 0 0 0
'1'	1	0 0 0 L
'2'	2	0 0 L 0
'3'	3	0 0 L L
'4'	4	0 L 0 0
'5'	5	0 L 0 L
'6'	6	0 L L 0
'7'	7	0 L L L
'8'	8	L 0 0 0
'9'	9	L 0 0 L
'A'	10	L 0 L 0
'B'	11	L 0 L L
'C'	12	L L 0 0
'D'	13	L L 0 L
'E'	14	L L L 0
'F'	15	L L L L

Bild 4.2 Sedezimalziffern

Die Ganzadresse (16 Bits = 4 Tetraden) kann maximal durch 4 Sedezimalziffern bis 'FFFF' dargestellt werden. Die den Sedezimalziffern entsprechenden Bitmuster werden bei der Übersetzung rechtsbündig in den Ganzadressenteil eingetragen.

Extern: Tetradenfolge

Intern: Bitmuster

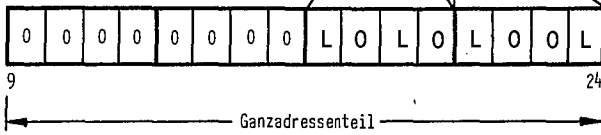


Bild 4.3 Tetradenfolge in einer Ganzadresse

Für Tetradenfolgen ist folgende Form niederzuschreiben:

't' t: Tetradenfolgen, durch Sedezimalziffern 0 - F dargestellt

Bei der externen Darstellung werden Tetradenfolgen zwischen Apostrophen eingeschlossen. Es wird nicht jede einzelne Sedezimalziffer, sondern die ganze Tetradenfolge eingeschlossen. Verwechslungen mit Dezimalzahlen werden durch die einschließenden Apostrophe ausgeschlossen.

Um bei der externen Darstellung von Tetradenfolgen eine übersichtliche Gliederung zu erzielen ist es erlaubt, die Sedezimalziffern durch Zwischenräume zu trennen. Zwischenräume werden nicht übersetzt und haben keinen Einfluß auf das Bitmuster in der Ganzadresse.

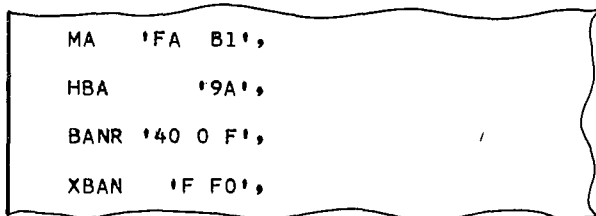


Bild 4.4 Beispiele für Tetradenfolgen in Ganzadressen

4.3. Oktaden- und Textfolgen

Absolute Angaben können in Ganzadressen auch mit Hilfe von Oktaden- und Textfolgen gemacht werden.

Oktadenfolgen bestehen aus einem oder mehreren Zeichen. Jedes Zeichen repräsentiert eine Kombination von 8 Bits (Oktade) und hat dezimal einen Wert zwischen 0 und 255.

Die Zuordnung der Zeichen zu den entsprechenden Oktaden ist festgelegt und aus der Tafel Zentralcode zu ersehen.

In einer Ganzadresse läßt sich eine Oktadenfolge durch zwei Zeichen darstellen. Eingeben lassen sich durch eine Oktadenfolge nur Zeichen, die auf den Eingabegeräten darstellbar sind.

Für Oktadenfolgen ist folgende Form der Niederschrift festgelegt:

'k' k: Oktadenfolge, durch Zeichen gemäß dem Zentralcode festgelegt

Bei der externen Darstellung werden Oktadenfolgen durch je zwei Apostrophzeichen eingeschlossen. Es ist zu beachten, daß zwei einzelne Apostrophzeichen hintereinander eingegeben werden und nicht etwa ein Anführungszeichen. Innerhalb der Oktadenfolge dürfen nur noch einzelne Apostrophzeichen dargestellt werden (um die Abgrenzung der Oktadenfolge nicht aufzuheben). Die Zeichen einer Oktadenfolge dürfen bei der Niederschrift nicht durch Zwischenräume getrennt sein. Zwischenräume stehen für sich selbst und werden vom Assembler entsprechend dem Zentralcode in das Bitmuster 'AF' übersetzt.

Wird extern in einer Oktadenfolge nur ein Zeichen dargestellt, so werden die rechten 8 Bits der Ganzadresse besetzt.

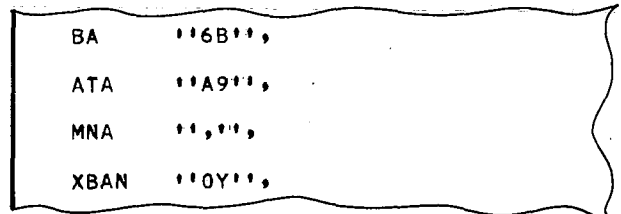


Bild 4.5 Beispiele von Oktadenfolgen in einer Ganzadresse

Textfolgen bieten die Möglichkeit, solche Zeichen des Zentralcodes darzustellen, denen keine Eingabezeichen zugeordnet sind. In Form einer Textfolge können diese Zeichen durch ihren dezimalen Wert und durch ihre entsprechenden Tetradenfolgen dargestellt werden. Ferner können mit Hilfe des Befehls TEXT Namen für Zeichen vereinbart und in einer Textfolge angegeben werden (siehe Abschnitt "Konstanten", 1.6.)

Die externe Darstellung hat folgende Form:

'(b)' b: Elemente (mehrere durch Komma getrennt)
 Dezimalwert der Oktade 0...255
 Sedezimalwert der Oktade '00'...'FF'
 Name der Oktade
 Oktadenfolge

Textfolgen werden jeweils eingeleitet und abgeschlossen durch die beiden Zeichen '()'. Innerhalb der Textfolge werden die einzelnen Elemente (Darstellungsformen) durch Kommas getrennt.

Die Elemente werden vom Assembler in die entsprechenden Bitmuster übersetzt und rechtsbündig in die Ganzadresse eingetragen.

```

BA      '(VW)',
BAR     '(152,'AF')',
E=      TEXT  218,
      └──┬──┘
          |
          v
BA      '(E,'AF')',
  
```

Bild 4.6 Beispiele von Textfolgen in Ganzadressen

4.5. Absolute Summenangaben

Werden in Ganzadressen absolute Angaben durch Plus- oder Minuszeichen verbunden, so bildet der Assembler die Summen dieser Angaben und setzt diese Summen in den Ganzadressenteil des betreffenden Befehls. Es ist möglich, alle im Abschnitt 4. beschriebenen Formen der Darstellung für absolute Summenangaben zu verwenden. Auch die in Abschnitt 4.4. beschriebenen Differenzangaben können Verwendung finden. Ebenso können zur Summenbildung Indexnamen benutzt werden (siehe Abschnitt 4.6. und 6.).

Das Ergebnis der Summenbildung darf den erlaubten Bereich für den Befehl nicht überschreiten.

```

A=      XBA  '8000'+120,
        BA  ''Q''-'F'+2,
C=      XC  X1,
        AA  X1+'A',
        SBA '(('E'', 'AF')'-X1,
        ETA C-A+12,
B=      XBA C-A+'('E'', 'AF')',
        MA  12+A-C,
D=      AA  '(12,'A0')'+X1,
        BA  B-A+B-D,
  
```

Bild 4.8 Beispiele für die Schreibweise von Summenangaben

4.4. Differenzangaben

In Ganzadressen können mit Hilfe von Kernspeicherbezügen Namen angegeben werden. Diese Namen symbolisieren Kernspeicheradressen, denen erst bei späterer Montage eine absolute Kernspeicheradresse zugeteilt wird (siehe Abschnitt 5.).

Der Assembler kann aber schon bei der Übersetzung die Differenz zwischen verschiedenen Namen ermitteln und daraus absolute Angaben bilden. Bei der Niederschrift für ein Programm werden die Namen mit Minuszeichen verbunden. Es ist zu beachten, dass Adressen, zwischen denen eine Differenz ermittelt werden soll, in der gleichen Zone definiert sind (siehe Abschnitt "AUFBAU").

```

ANF=    XBA  100/V,
E=      AA  END-E+20,
        ST  END-E+12 A1,
        HBC '10'+END-E,
END=    TBC  +3R-ANF,
  
```

Bild 4.7 Beispiele für die Schreibweise von Differenzangaben

4.6. Symbolische Indexadressen

An Stelle von Dezimalzahlen oder Sedezimalziffern, die Indexzellen absolut adressieren, können Namen zur symbolischen Adressierung von Indexzellen Verwendung finden. Die Namen müssen stets mit einem Buchstaben beginnen. Diesem Buchstaben können weitere Buchstaben und Ziffern folgen. Die Länge des Namens ist nicht begrenzt, er darf aber auch nur aus einem Buchstaben bestehen.

Indexnamen müssen nicht besonders (explizit) definiert werden. In Abhängigkeit vom jeweiligen Befehlscode werden sie (implizit) im Adressenteil des Befehls erwartet und eindeutig als Indexadressen definiert.

Der Assembler übersetzt die symbolischen Namen in die absoluten Adressen von Indexzellen, und zwar in der Reihenfolge ihres (statischen) Auftretens. So wird also dem zuerst auftretenden Namen die Indexzelle 0 zugeordnet, den folgenden Namen 1, 2, 3, usw. (siehe Bild 4.8). Maximal können so 256 Indexzellen (0 - 255) adressiert werden. Den Indexzellen werden bei der Übersetzung absolute Adressen zugeordnet, die bei späterer Montage nicht mehr verändert werden (siehe auch Abschnitt 6.).

Werden im Adressenteil eines Befehls zwei Indexadressen dargestellt (i_L, i_R), so übersetzt der Assembler zuerst die linke und danach die rechte Adresse. Für Namen, die mit Hilfe des Pseudobefehls INDEX vereinbart wurden, gilt eine andere Form der Adressenzuordnung (siehe Abschnitt 6.2.).

Wurden Indexadressen absolut durch Ziffern angegeben und ist die entsprechende Adresse schon vorher einem symbolischen Namen zugeordnet worden, so wird dies nicht mehr geändert. Das bedeutet z.B., daß dem zuerst auftretenden Namen XANF die Indexadresse 0 zugeordnet wird; folgte danach bei einem Befehl die Adressierung

mit der absoluten Zahl 0 (z.B. XC 0), so würde die gleiche Indexzelle angesprochen.

Eine ausführliche Beschreibung der Indexadressierung ist im Abschnitt 6. geboten.

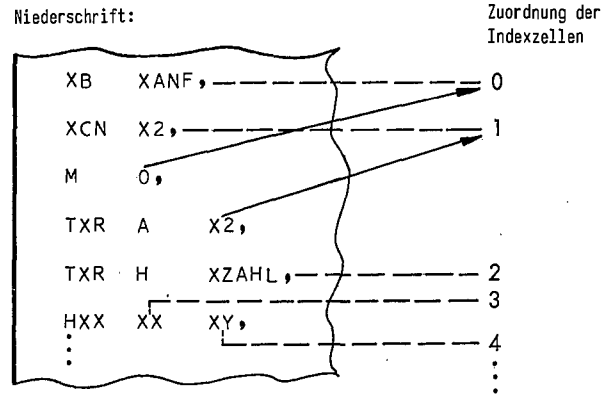


Bild 4.9 Beispiele Indexadressen und Zuordnung

Eine große Anzahl von TR 440-Befehlen verlangt in den Adressenteilen die Angabe bestimmter Kernspeicheradressen. In den Schriften "TR 440 Befehls-Lexikon" und "TR 440 Große Befehlsliste" ist bei diesen Befehlen in den Spalten Adressenteil (adr) ein n für die Adressen von Ganzwörtern oder ein m für die Adressen von Halbwörtern eingetragen. In bestimmten Fällen können auch Befehle, die in dieser Spalte ein z (Operand) eingetragen haben, zu dieser Gruppe gezählt werden (siehe Abschnitt 2.3.). Die Adressen der jeweiligen Kernspeicherzellen lassen sich bei diesen Befehlen durch symbolische Namen (Kernspeicherbezüge) darstellen. Diese Namen müssen vorher für die bestimmten Kernspeicherzellen vereinbart werden. Diese Form der Kernspeicheradressierung ist von Vorteil, da sich in vielen Fällen absolute Adressen schwer angeben lassen, da es oft unbekannt sein wird, unter welchen Kernspeicheradressen Teile eines Programms abgelegt werden.

Bei der Übersetzung werden Kernspeicherbezüge vom Assembler zuerst in eine zur jeweiligen Adressenzone relative Adresse übersetzt. Erst bei der späteren Montage erfolgt die Umwandlung der relativen Adressen durch Addition von Translationsadressen in absolute Adressen.

Der Assembler prüft, ob die angegebenen Kernspeicherbezüge den Konventionen entsprechen und somit zulässig sind (nicht zulässig sind z.B. Abspeicherungen in schreibgeschützte Bereiche).

Adressen können eine Länge von 16 Bits annehmen, damit können 65 536 Halbwörter bzw. 32 678 Ganzwörter adressiert werden. Bei Befehlen, die Operanden aus dem Speicher holen oder in den Speicher bringen, liegt das adressierte Wort in der Großseite 0 (K- oder V-Bereich). Bei einem großen Teil der Sprungbefehle wird ein Wort in der Großseite 1 (B-Bereich) adressiert.

5.1. Symbolische Adressen

Wird im Adressenteil eines Befehls (statt einer Adresse) ein Name angeführt, so steht dieser symbolische Name für die Adresse der Kernspeicherzelle, die mit diesem Namen benannt wurde. Der Name wurde im Benennungsteil der Informationseinheit einem Informationsteil vorangestellt (siehe Bild 1.1 sowie den Abschnitt "Aufbau")

Ein symbolischer Name muß mit einem Buchstaben beginnen (Umlaute Ä, Ö und Ü sind ausgeschlossen); er darf allein aus diesem ersten Buchstaben bestehen oder aus einer Folge von weiteren Buchstaben und Ziffern zusammengesetzt sein. Erlaubt ist eine Folge von maximal 31 Zeichen.

5.1.1. Symbolische Kernspeicheradressen im Adressenteil von Befehlen

Der Adressenteil eines TR 440-Befehls ist 16 Bits lang. Damit ist es möglich, maximal 65 536 Halbwörter oder 32 768 Ganzwörter zu adressieren. Wird im Adressenteil eines Befehls ein symbolischer Name für eine Kernspeicherzelle angegeben, so ist vorauszusetzen, daß dieser Name vorher für eine bestimmte Speicherzelle vereinbart wurde.

Befehle, die Operanden aus dem Speicher holen, benötigen in ihrem Adressenteil Namen, welche Adressen in der Großseite 0 (K- oder V-Bereich) symbolisieren (siehe Abschnitt "Aufbau", 3. Gliederung des Adressenraumes).

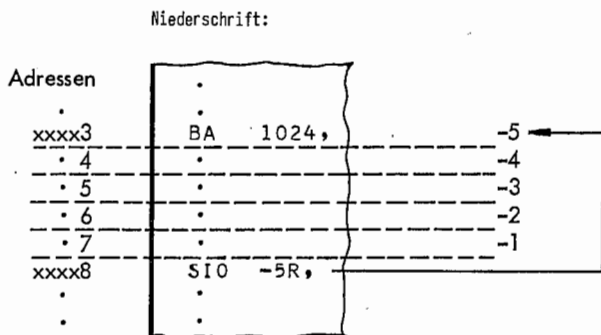
Symbolische Kernspeicheradressen in den Adressenteilen von Sprungbefehlen werden im Abschnitt 5.1.2. beschrieben.

Der Assembler übersetzt die Kernspeicherbezüge in relativ zur Adressenzone stehende Adressen. Die Umwandlung in absolute Adressen kann erst bei der Montage erfolgen.

Werden bestimmte Größen (z.B. Konstanten) mit einem symbolischen Namen entweder schreibgeschützt in den K-Bereich oder nicht schreibgeschützt in den V-Bereich gebracht, so sind diese Speicherzellen mit 16- und 22-Bit-Adressen adressierbar.

Zur Ausführung wird der Befehl in das Befehlswerk geholt. Während der Abruflphase wird der Adressenteil des

übersetzt werden kann. Mit Relativbezügen lassen sich Namen einsparen.



nach Assemblierung:

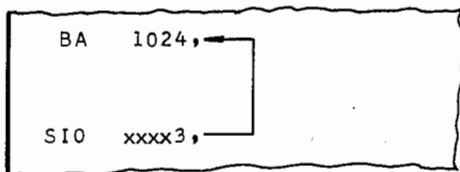


Bild 5.3 Relative Adresse bei einem Sprungbefehl

Es ist darauf zu achten, daß der Abstand zwischen Sprung und Ziel nicht mehr durch eine Korrektur des Programms (durch dazwischengesetzte oder entfernte Befehle) verändert wird.

Siehe hierzu auch Abschnitt "Sonstige Operationen", 2.1.2.

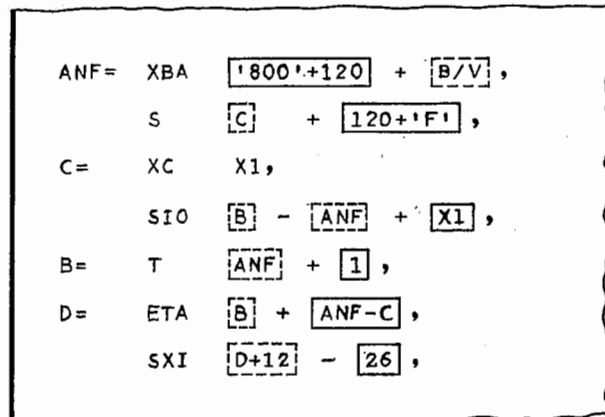
Summenadressen können aus beliebig vielen miteinander verknüpften Gliedern bestehen. Die einzelnen Glieder können in folgenden Formen auftreten:

- | | | |
|----------------------------|---|--------------------|
| Kernspeicheradressen-Namen | } | Kernspeicherbezüge |
| Summenadressen | | |
| absolute Angaben | } | absolute Bezüge |
| Differenzangaben | | |
| Absolute Summenangaben | | |
| Indexnamen | | |

Es sind aber nur Kernspeicherbezüge mit absoluten Bezügen verknüpfbar. (Kernspeicher-Bezug + Kernspeicherbezug, sowie absoluter Bezug + absoluter Bezug werden nicht als Summenadressen in diesem Sinne betrachtet).

Die in den einzelnen Abschnitten beschriebenen Formen der absoluten Angaben und der symbolischen Kernspeicheradressierung (Abschnitte 4, bis 5.2.) dürfen Glieder der Summenadresse sein und miteinander kombiniert werden. Das erste Glied darf kein Vorzeichen haben. Alle weiteren Glieder werden durch Vorzeichen zur Summenbildung verknüpft.

Niederschrift:



- | | | |
|--|-----------------------|-------------------|
| | = absolute Angaben | } im Adressenteil |
| | = Kernspeicher-Bezüge | |

Bild 5.4 Beispiele von Summenadressen

5.3. Summandenadressen

Zur Adressierung von Kernspeicherzellen können Kombinationen von Kernspeicher- und absoluten Bezügen (die durch Plus oder Minus verknüpft werden) Verwendung finden. Diese Kombinationsmöglichkeit ist vielfältig und wird als Summenadresse bezeichnet. Eine Summenadresse in dieser Kombinationsform wird als Kernspeicherbezug interpretiert und vom Assembler erst bei der Montage in eine absolute Adresse verwandelt.

Indexzellen haben Halbwortlänge und stellen somit eine 24-Bitgröße dar. Sie können damit alle Speicheradressen aufnehmen. Die Indexzellen liegen hintereinander im Speicher. Sie bilden die Indexzone. Die Indexzone kann bis zu 256 Halbwörter lang sein.

Den Indexzellen sind die Indexadressen zugeordnet. Dabei hat die erste Indexzelle die Adresse 0, und die maximale Indexadresse ist 255.

Die Indexzellen können durch einen Befehl angesprochen werden, der im Adressenteil den Buchstaben i hat. Für den Buchstaben i ist die Adresse der Indexzelle anzugeben.

Die Lage der Indexzone innerhalb des Speicherbereichs wird dadurch festgelegt, daß die Speicheradresse des ersten Halbwortes, das ist die Indexzelle mit der Indexadresse 0, in das Register X (Indexbasisregister) gebracht wird.

Vor dem Start eines Programms (Operatorlauf) muß das Register X (Indexbasisregister) gesetzt werden. Hierzu ist der Pseudobefehl XBASIS vorhanden. Während des Laufs kann das Register X mit Hilfe der Befehle **BCI** und **ZI** umgesetzt und damit eine andere Indexzone festgelegt werden.

In Bild 6.1 ist an einem Beispiel die Lage der Indexzone im Kernspeicher gezeigt. Das graue Feld ist die Indexzone. Sie besteht aus 12 Halbwörtern. Der Anfang der Indexzone ist durch die Speicheradresse n+1 festgelegt. Diese Adresse steht im Register X. Die Adresse n+13 wird bereits wieder als Speicherzelle (Halbwort) verwendet, wenn für die Indexzone nur 12 Halbwörter freigehalten werden.

Die Indexzellen können auch über Speicheradressen angesprochen werden, wenn die Indexbasisadresse bekannt ist. (Dies ist die Speicheradresse des Halbwortes, unter dem die Indexzelle 0 liegt.) Hierbei muß jedoch beachtet werden, daß die 4 zuletzt benutzten Indexzellen im Indexregister des Befehlswerks stehen und dort auch bleiben und verändert werden können. Wird die Indexzelle über ihre Speicheradresse angesprochen, so kann es sein, daß dort nicht der aktuelle Wert steht und daß die Zelle nachher mit ihrem aktuellen Wert überschrieben wird. Daher ist es im allgemeinen nicht zweckmäßig, Indexzellen über Speicheradressen anzusprechen.

6.1. Indexadressen

Dort wo in der Beschreibung der Befehle im Adressenteil der Buchstabe i angegeben ist, muß die Adresse einer Indexzelle angegeben werden. Es stehen 256 Indexzellen zur Verfügung, so daß die internen Indexadressen in 8 Binärstellen dargestellt werden. Bei der Niederschrift des Programms (externe Darstellung) können die Indexadressen, dezimal, sedezimal oder symbolisch angegeben werden. Des weiteren ist es möglich, eine Summenadresse zu bilden.

6.1.1. Absolute Indexadressen

Absolute Indexadressen können als Dezimalzahl oder als Sedezimalzahl geschrieben werden. Die erste Indexadresse hat den Wert 0 - dezimale Indexadressen können also die Werte 0 bis 255 annehmen.

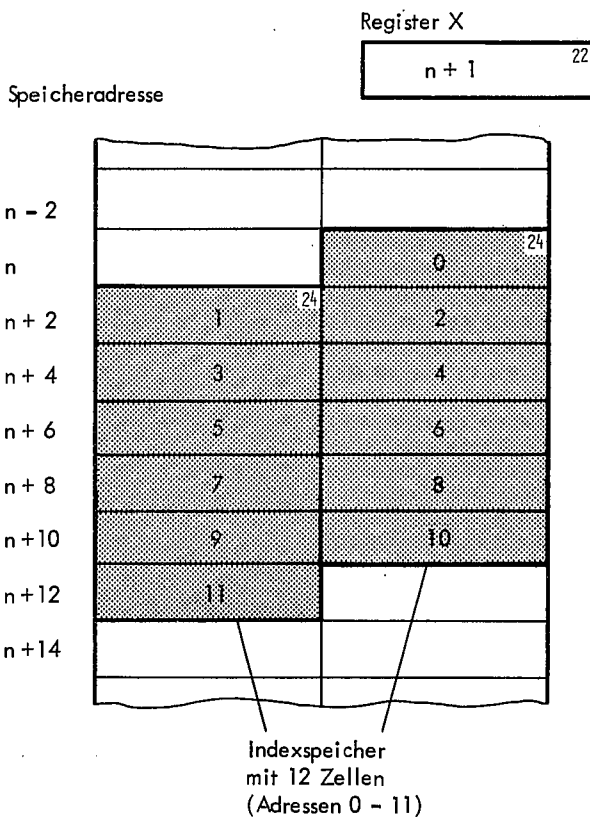


Bild 6.1 Beispiel für Indexspeicher

XB	15,
XC	8,
E	B '0B',
ZX	10 'A',
TTX	'F0' 120,

Bild 6.2 Beispiele für absolute Indexadressen

TAS - Handbuch

Nov. 68

Für die sedezimale Schreibweise werden die Ziffern 0 bis 9 und die Buchstaben A bis F verwendet. Die sedezimale Indexadresse kann die Werte '0' bis 'FF' annehmen, wobei die sedezimale Adresse 'FF' der dezimalen Adresse 255 entspricht. Sedezimale Adressen müssen stets in Apostrophe eingeschlossen werden. Wie bei Dezimalzahlen können führende Nullen entfallen ('0B' oder 'B').

6.1.2. Symbolische Indexadressen

Alle Indexzellen können durch eine oder mehrere Adressen adressiert werden. Die symbolische Adresse wird durch einen Namen dargestellt. Namen beginnen stets mit einem Buchstaben. Es können gemischt Buchstaben und Zahlen folgen. Die Länge der Namen ist nicht begrenzt.

Indexnamen brauchen nicht besonders (explizit) definiert zu werden. Durch ihr erstmaliges Auftreten im Adressenteil eines Befehls werden sie (implizit) definiert. Beim Assemblieren des Programms werden den symbolischen Indexadressen in der Reihenfolge ihres (statischen) Auftretens absolute Adressen zugeordnet. Die erste symbolische Indexadresse erhält die absolute Adresse 0, die folgende 1 usw. Enthält ein Befehl zwei symbolische Indexadressen, so wird zuerst die linke und danach die rechte zugeordnet. Wird der Pseudobefehl INDEX verwendet, so gilt eine andere Zuordnung (siehe Abschnitt 6.2).

Alle absolut angegebenen Indexadressen bleiben bei dieser Zuordnung unberücksichtigt; es wird also nicht geprüft, ob eine Indexzelle bereits durch die Angabe einer absoluten Adresse belegt wurde. Beginnt z.B. ein Programm mit der Befehlsfolge

```
ZX  -10 X1,
XB   0,
```

so wird der symbolischen Indexadresse X1 die absolute Indexadresse 0 zugeordnet. Dies ist die gleiche Indexzelle, wie die mit dem Befehl XB 0 adressierte.

HXX	X1	X2,
XB	X3,	
MH	5	ANZ,
ZX	10	B,
M	X3ANTON,	

Bild 6.3 Beispiele für symbolische Indexadressen

Alle symbolischen Indexadressen, die implizit definiert werden, haben einen globalen Geltungsbereich, gelten also auch über die Segmentgrenzen hinweg. Bei den durch den Pseudobefehl INDEX explizit definierten symbolischen Indexadressen gelten andere Bedingungen. (s. dazu Abschnitt 6.2.5).

Ein Name darf nicht gleichzeitig als symbolische Adresse für eine Indexzelle und für eine Speicheradresse verwendet werden, da bei Summenadressen Index- und Speicheradressen gemischt auftreten dürfen. Zugelassen ist es jedoch, wenn die Gültigkeitsbereiche zweier gleichlautender Namen sich nicht überschneiden, wenn z. B. beides lokale Namen sind und in verschiedenen Segmenten liegen.

6.1.3. Summenadressen

Eine Indexadresse kann auch aus einer Summe von mehreren Adressen bestehen. Dieser Summenbezug kann beliebig viele Glieder haben, die beliebig durch Plus- oder Minus-Zeichen verbunden sind. Die einzelnen Glieder können sowohl Indexadressen als auch Speicheradressen sein. Des Weiteren sind Zahlen in dezimaler oder sedezimaler Form erlaubt. Das erste Glied darf kein Vorzeichen haben. Nach der Summierung muß der Wert der Adresse positiv sein und darf nicht größer sein als 255.

In einem Summenbezug werden alle nicht anderweitig definierten symbolischen Adressen als implizit definierte Indexadressen aufgefaßt.

ZX	-10	A-B+3,
EZ	C	X1+*F',
MD	*A'+B	A+*F0'-20,

Bild 6.4 Beispiele für Summenadressen

6.2. Pseudobefehl INDEX

Mit Hilfe des Pseudobefehls INDEX kann dem Assembler die Anweisung gegeben werden, symbolischen Indexadressen bestimmte absolute Indexadressen zuzuordnen. Der Befehl hat die Form

INDEX p (x₁, ..., x_n),
 p: Indexpegel, absolute Indexadresse, symbolische Indexadresse oder leer

x: Indexliste (1 bis n Glieder)
 Indexname oder Freihaltezahl (0 bis 255)

Der Befehl INDEX bedeutet gleichzeitig eine explizite Festlegung von Indexnamen. Während bei der impliziten Angabe von symbolischen Indexadressen diese Adressen stets globalen Geltungsbereich haben, also auch über die Segmentgrenzen hinaus Geltung hatten, können bei dieser expliziten Festlegung die symbolischen Indexadressen sowohl einen globalen, als auch einen lokalen Geltungsbereich haben (siehe Abschnitt 6.2.5.).

Beim Assemblieren werden zuerst die durch die Pseudobefehle INDEX angegebenen Indexzellen belegt. Allen symbolischen Indexadressen, die nicht durch den Pseudobefehl INDEX explizit festgelegt wurden - die also nur implizit festgelegt sind - werden absolute Indexadressen hinter der höchsten durch den Pseudobefehl INDEX vergebenen absoluten Indexadresse zugeordnet. Absolute Indexadressen in Befehlen benötigen keine Zuordnung und werden bei der Zuordnung auch nicht berücksichtigt. Ist die explizite Zuordnung nicht lückenlos, so bleiben die Indexzellen unbenutzt (siehe auch Beispiel in Bild 6.5).

6.2.1. Absolute Pegelangabe

Mit dem Pseudobefehl INDEX ist es möglich, eine feste Zuordnung von symbolischen zu absoluten Adressen vorzugeben. Hinter dem Pseudobefehl INDEX wird angegeben, ab welcher absoluten Adresse mit der Zuordnung begonnen werden soll. Dies ist der Indexpegel. Danach wird - in Klammern eingeschlossen und durch Komma getrennt - eine oder mehrere symbolische Indexadressen aufgeführt. Die erste symbolische Adresse erhält die durch den Indexpegel angegebene Adresse, die zweite die nachfolgende, usw.

Niederschrift

```

SEG,
INDEX 20 (X1,X5,X4),
INDEX 0 (X2,ANZ),
INDEX 1 (X3),
TTX A1 A2,

```

Zuordnung

Adresse	
symbolisch	absolut
X2	0
X3, ANZ	1
X1	20
X5	21
X4	22
A1	23
A2	24

Bild 6.5 Beispiel für absoluten Indexpegel

Im Bild 6.5 ist ein Beispiel angegeben. Sind im Programm keine weiteren "INDEX"-Befehle, so werden der ersten implizit angegebenen Indexzelle die Adresse 23, der nächsten die Adresse 24 usw. zugeordnet.

Die Indexliste darf keine Summenadressen enthalten. Die mit diesem Befehl angegebene Zuordnung erfolgt ohne Rücksicht auf irgendwelche andere Zuordnungen und ohne Rücksicht auf Segmentgrenzen und eingestellte Indexbasis.

6.2.2. Symbolische Pegeladressen

Anstelle der absoluten Pegelangabe, wie im vorhergehenden Abschnitt beschrieben, kann auch eine symbolische Adresse als Indexpegel angegeben werden. Die erste symbolische Adresse in der Indexliste erhält dann die gleiche absolute Adresse, wie die als Indexpegel angegebene symbolische Adresse.

Die symbolische Pegeladresse muß vorher explizit, d. h. also mit Hilfe eines Befehls INDEX definiert sein. Außerdem muß sie noch gültig sein, d. h. wenn sie in einem vorhergehenden Segment (explizit) definiert wurde, muß sie einen globalen Geltungsbereich haben (Geltungsbereich, siehe Abschnitt 6.2.5.).

Mit Hilfe eines symbolischen Indexpegels ist es möglich, eine Indexzelle mit mehreren Namen zu belegen, bzw. die Indexzelle in einem anderen Segment mit einem anderen Namen zu versehen.

Niederschrift

```

S1= SEG,
    :
    :
    INDEX 0 (ANZ.,X1.,X2),
    :
    :
S2= SEG,
    :
    :
    INDEX X1 (A,B),
    :
    :
S3= SEG,
    :
    :
    INDEX X1 (2,X.,Y),
    :
    :

```

Zuordnung

absolute Adresse	Segment S1	Segment S2	Segment S3
0	ANZ	ANZ	ANZ
1	X1	X1,A	X1
2	X2	B	frei
3	X	X	X
4			Y

Bild 6.6 Beispiel für symbolischen Indexpegel

6.2.3: Reihenfolge festlegen

Die Angabe eines Indexpegels kann auch fehlen. In diesem Fall wird bei der ersten Indexdeklaration des Programms mit der Zuordnung ab der Indexadresse 0 begonnen, als ob der Pegel 0 stände.

Folgen auf Indexdeklarationen mit Pegelangabe weitere Indexdeklarationen, bei denen die Pegelangabe fehlt, so wird in diesem Fall der Pseudobefehl INDEX als Verlängerung des letzten Pseudobefehls INDEX betrachtet. Dies ist in Bild 6.7 aus der Tabelle zum Programmbeispiel ersichtlich.

Im Beispiel Bild 6.7 ist der zweite und dritte Pseudobefehl INDEX eine Fortsetzung des ersten. Der erste INDEX-Befehl im Segment S3 ist eine Fortsetzung des zweiten INDEX-Befehls aus Segment S2. Von den ersten symbolischen Indexadressen im Befehl TXX (Segment S3) wurde X1 bereits explizit festgelegt. Die Indexadresse X7 ist eine implizit festgelegte und gilt damit global in allen Segmenten.

Niederschrift

```

S1=  SEGM,
      INDEX 0(X1,X2,X3),
      INDEX (X4,X5,X6),
      .
      .
      .
S2=  SEGM,
      INDEX (X1,X2,X3),
      INDEX 0(X4,X5,X6),
      .
      .
      .
S3=  SEGM,
      TXX X1 X7,
      INDEX (X1,X2,X3),
      INDEX 0(X4,X5,X6)
      .
      .
  
```

Zuordnung

absolute Adresse	Segment S1	Segment S2	Segment S3
0	X1	X4	X4
1	X2	X5	X5
2	X3	X6	X6
3	X4		X1
4	X5		X2
5	X6		X3
6		X1	
7		X2	
8		X3	
9	X7	X7	X7

Bild 6.7 Beispiele für Pseudobefehle INDEX ohne Pegelangabe

6.2.4: Freihalteanweisung

Mit Hilfe des Befehls INDEX ist es möglich, Indexzellen freizuhalten, ohne ihnen Namen zuzuordnen. Anstelle der Indexadresse wird in der Indexliste eine Dezimalzahl angegeben. Sie gibt an, wieviel Indexzellen freigehalten werden sollen. Die Freihaltezahl kann die Werte 0 bis 255 annehmen.

Die freigehaltenen Indexzellen können über Summenadressen, durch Modifizierung oder durch spätere explizite Benennung mittels des Pseudobefehls INDEX angesprochen werden.

Niederschrift

```

S1=  SEGM,
      .
      .
      INDEX 16 (X1,,3,ANZ,A,B,.)
      .
      .
      XB X1+2,
      TTX X1+1 H,
      MCF (2/H),
      XB X1,
      .
      .
      .
S2=  SEGM,
      .
      .
      TXX X Y,
      .
      .
      INDEX X1 (1,X,Y,Z),
      .
      .
      .
S3=  SEGM,
      .
      .
      INDEX X1(1,U,V,W),
      .
      .
      .
      TTX U V,
      .
      .
  
```

Zuordnung

absolute Adresse	Segment S1	Segment S2	Segment S3
16	X1	X1	X1
17	frei	X	U
18	frei	Y	V
19	frei	Z	W
20	ANZ	frei	frei
21	A	frei	frei
22	B	B	B
23	H	H	H

Bild 6.8 Beispiel für Freihalteanweisung

6.2.5. Geltungsbereich

Während alle implizit festgelegten Indexzellen (gemäß Abschnitt 6.1.) einen globalen Geltungsbereich haben, d. h. innerhalb des durch den ersten Pseudobefehl SEGM und den Pseudobefehl ENDE festgelegten Programms oder Quellenprogramms gelten, ohne Rücksicht auf Segmentgrenzen, kann bei der expliziten Festlegung der Indexzellen durch den Pseudobefehl INDEX zwischen globalen und lokalem Geltungsbereich gewählt werden.

Die in der Indexliste beim Pseudobefehl INDEX aufgeführten Indexadressen gelten nur in dem Segment, in dem der Pseudobefehl INDEX steht. Sollen sie jedoch einen globalen Geltungsbereich haben, so müssen sie (wie bei der Benennung von Speicherzellen) mit einem Punkt versehen werden.

Mit dem Pseudobefehl INDEX besitzt man damit die Möglichkeit, Indexnamen festzulegen, die nur in einem Segment gültig sind.

Niederschrift

```

S1=  SEGM,
      INDEX 0 (X1.,A,B,C.,D),
      .
      .
      .
S2=  SEGM,
      .
      .
      .
      INDEX X1 (1,H,I,J,K),
      .
      .
      .
    
```

Zuordnung

absolute Adresse	Segment S1	Segment S2
0	X1	X1
1	A	H
2	B	I
3	C	C,J
4	D	K

Bild 6.9 Beispiel für lokale Indexadressen

Während des Laufs des Programms kann das Register X mit Hilfe der Befehle BCI und Z1 auf einen anderen Wert gesetzt werden. Damit ist es möglich, die Indexzone zu wechseln und auch die Anzahl der Indexzellen in einem Programm über 256 zu erhöhen.

Der Assembler nimmt die Belegung der Indexzellen stets nach dem gleichen Schema vor, unabhängig von der eingestellten Indexzone und auch unabhängig davon, ob auf eine andere Zone umgeschaltet wird (weiteres siehe Abschnitt 6.3.5.).

In der internen Darstellung enthält das Indexbasisregister (Register X) die Kacheladresse der ersten Indexzelle, das ist die Adresse, wie sie intern dem Speicher übergeben wird. Sie ist also bereits über die Seitenadressenregister ersetzt. Dies ist jedoch für die Programmierung bedeutungslos.

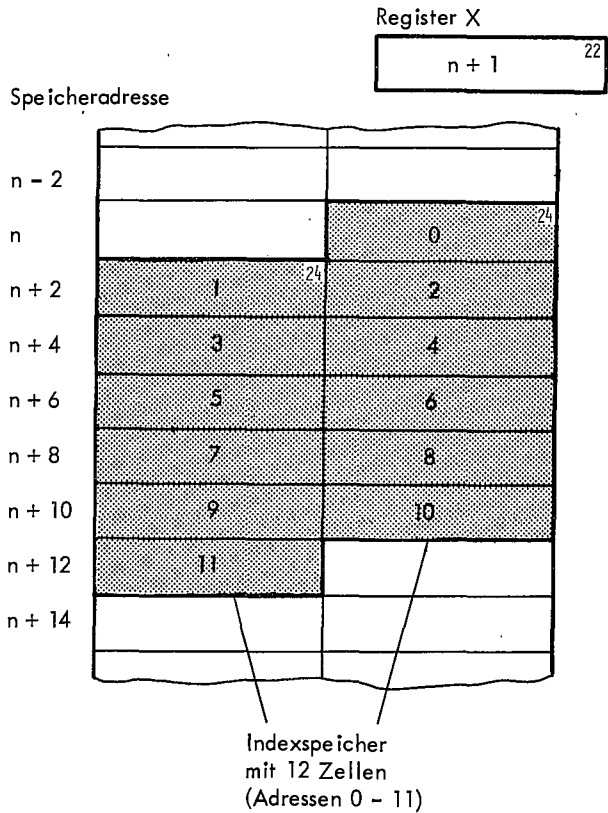


Bild 6.10 Beispiel für Indexspeicher

6.3. Indexbasis

Die Indexbasisadresse adressiert die Speicherzelle (Halbwort), die die Indexzelle mit der Indexadresse 0 enthält. Die Indexbasisadresse steht im Register X (Indexbasisregister). Das Register X muß vor dem Start eines Programms (Operatorlauf) gesetzt werden. Dazu dient der Pseudobefehl XBASIS. Er legt die Lage der Indexzone im Kernspeicher fest. Gleichzeitig muß mit dem Pseudobefehl ASP oder DSP die Länge der Indexzone (Anzahl der Indexzellen) festgelegt werden (siehe Abschnitt 5.3.1).

6.3.1. Grundeinstellung

Bevor ein Programm gestartet wird (vor dem Operatorlauf) müssen bestimmte Voreinstellungen getroffen werden. Diese Voreinstellungen werden mit Hilfe von Pseudobefehlen am Anfang des Programms bzw. am Anfang des ersten Quellenprogramms angegeben. Dazu gehört unter anderem das Setzen des Registers X (Indexbasisregister). Dies geschieht mit dem Pseudobefehl XBASIS.

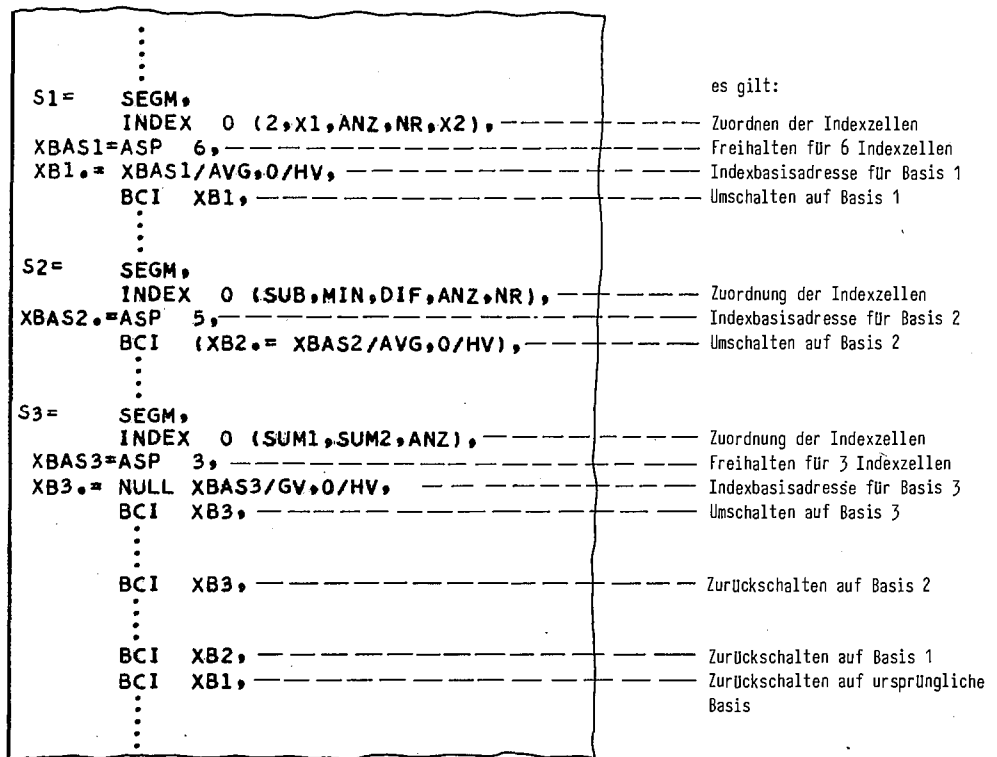


Bild 6.13 Beispiel für mehrfache Indexbasisumschaltung

fehl BCI wird nun der Inhalt dieses Ganzwortes mit dem Inhalt der Register X und U ausgetauscht. Mit einem zweiten Befehl BCI kann dies dann wieder rückgängig gemacht werden, d. h. der alte Zustand wieder hergestellt werden, oder aber es kann auf eine dritte Indexbasis umgeschaltet werden.

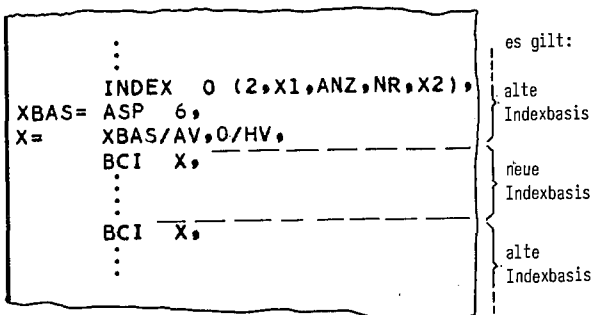


Bild 6.12 Umschalten der Indexbasis und zurückschalten

6.3.5. Indexadressen beim Umschalten

Eine Indexbasisumschaltung wird vom Assembler nicht berücksichtigt, da ihm der dynamische Ablauf des Programms nicht bekannt ist. Er ordnet also, wie in den vorstehenden Abschnitten 6.1 und 6.2 beschrieben, die absoluten Indexadressen den symbolischen Indexadressen zu.

Der Programmierer muß also, da er den dynamischen Ablauf kennt, für jede eingestellte Indexbasis die gewünschte Zuordnung festlegen.

Eine Möglichkeit besteht darin, nur absolute Indexadressen zu verwenden oder aber in einer Indexbasis beliebige Indexadressen und in den weiteren Indexbasen nur absolute Indexadressen.

Das gleiche kann erreicht werden, wenn symbolische Indexadressen verwendet werden, diese aber mit Hilfe des Pseudobefehls INDEX mit Angabe eines absoluten Pegels für jede Indexbasis aufgelistet werden.

Für Unterprogramme kann es oft wünschenswert sein, daß sie eine eigene Indexbasis besitzen. Damit braucht der Indexzellenbedarf der Unterprogramme im Hauptprogramm nicht berücksichtigt zu werden. Einzelheiten dazu sind dem Abschnitt "Anschluß anderer Programme" unter Abschnitt 1 zu entnehmen.

Wird das Programm in mehrere Teile zerlegt und werden diese getrennt assembliert, so kann es zweckmäßig sein, diesen Quellenprogrammen eine eigene Indexbasis zuzuordnen. Es muß dabei im Quellenprogramm der Speicherbedarf für den Indexbereich freigehalten und die Indexbasis (Register X) auf die Anfangsadresse dieses Bereichs gesetzt werden. In diesem Sinn kann auch ein Unterprogramm, wenn es getrennt assembliert wird, als Quellenprogramm aufgefaßt werden.

Der Befehl BCI ermöglicht auf einfache Weise eine Zurückschaltung auf die alte Indexbasis. Wird mehrmals umgeschaltet, so wird am zweckmäßigsten in der Reihenfolge, in der auf die verschiedenen Indexbasen umgeschaltet wurde, auch wieder zurückgeschaltet. Dies kann

auch anders gemacht werden, nur muß dazu genau verfolgt werden, in welcher Zelle sich zur Zeit die Adresse der gewünschten Indexbasis befindet. Dies kann besser mit dem Befehl ZI erreicht werden. Der Befehl ZI berücksichtigt jedoch nicht das Unterprogrammregister. Dies könnte jedoch mit den Befehlen R B U, C2 XBAS+1 in einer Speicherzelle sichergestellt und mit MCF XBAS+1,ZU 0 auf den alten Wert zurückgesetzt werden. Des weiteren wäre eine Sicherstellung mit den Befehlen R TCB U, XC XU und ein Zurücksetzen mit dem Befehl E ZU XU, möglich.

Es muß in jedem Fall darauf geachtet werden, daß aus dem Bereich, in dem eine Indexbasis gilt, nicht in ein Programmteil gesprungen wird, in dem eine andere Indexbasis gilt und dort Indexzellen benutzt werden, ohne daß die Indexbasis auf den zugehörigen Wert gesetzt wird.

Quellenprogramm 1

```

TEIL1.=SEGM,
      XBASIS   X,
X=    ASP 256,
      ALARM   AADR,
      UNTPR   XO,
      EINGG   TEIL1,
AADR=
      .
      .
      .
      BAN 1,
      S   EING2,
      .
      EXTERN  TEIL2(EING2),
      .
      ENDE,
  
```

Quellenprogramm 2

```

TEIL2.=SEGM,
XBAS1= ASP 25/D,
X1 = XBAS1/AV,0/HV,
EING1.=BCI X1, ←
      .
      EINGG   TEIL2,
      EINGG   (EING1,EING2),
      .
      .
      S   +2R,
EING2.=BCI X1, ←
      .
      SK0 +3R,
      BCI X1, →
      S   TEIL1,
      .
      .
      BCI X1, →
      S   TEIL3,
      .
      .
      EXTERN  TEIL1,
      EXTERN  TEIL3,
      .
      ENDE,
  
```

Umsetzen der Indexbasis
bei Sprung nach Teil 2

Umsetzen der Indexbasis
bei Sprung nach EING2

Zurücksetzen der Indexbasis
wenn Rücksprung nach Teil 1

Zurücksetzen der Indexbasis
wenn Rücksprung nach Teil 3

Quellenprogramm 3

```

TEIL3.=SEGM,
      .
      EINGG   TEIL3,
      .
      ENDE,
  
```

Die anderen Quellenprogramme haben eine gemeinsame Indexbasis

Bild 6.14 Ein Quellenprogramm hat eigene Indexbasis

Quellenprogramm 1

```

TEIL1.=SEGM,
        XBAS1  XBAS0,
XBAS0= ASP 256,
        ALARM  AADR,
        UNTPR  X0,
        EINGG  TEIL1,
        EINGG  EING1,
        .
        S      +2R,
XBAS1= ASP 25/D,
EING1= ZI (XBAS1/A),
        .
        SGO   TEIL2,
        .
        S     TEIL3,
AADR=  .
        .
        EXTERN  TEIL2,
        EXTERN  TEIL3,
        .
        ENDE,
    
```

Quellenprogramm 2

```

TEIL2.=SEGM,
        ZI (NULL XBAS2/V),
XBAS2= ASP 5,
        .
        BAN  1,
        S    EING1,
        .
        EINGG  TEIL2,
        EXTERN  EING1,
        .
        ENDE,
    
```

Quellenprogramm 3

```

TEIL3.=SEGM,
XBAS3= ASP 5,
        ZI (XBAS3/A),
        .
        EINGG  TEIL3,
        .
        ENDE,
    
```

Bild 6.15 Jedes Quellenprogramm hat eigene Indexbasis

6.4. Indexadressen bei Quellenprogrammen

Werden Programme getrennt assembliert, so bezeichnen wir sie als Quellenprogramme. Quellenprogramme werden durch das Montageprogramm nach dem Assemblieren zusammengefügt.

Während die Speicheradressen bei der Montage der Quellenprogramme entsprechend translatiert werden, so daß sich die Speicheradressen der Quellenprogramme nicht überschneiden, bleiben die Indexadressen unverändert, d. h. die bei der Assemblierung festgelegten absoluten Indexadressen bleiben unverändert. Der Programmierer muß also dafür sorgen, daß sich die Indexadressen der verschiedenen Quellenprogramme nicht überschneiden.

6.4.1. Eigene Indexbasis

Der einfache Fall ist der, daß jedem Quellenprogramm eine eigene Indexbasis zugeordnet wird. Sprünge von einem Quellenprogramm in ein anderes müssen jedoch stets mit einer Umschaltung der Indexbasis auf das andere Quellenprogramm verbunden sein, wenn dort Indexzellen benutzt werden.

6.4.2. Gemeinsame Indexbasis

Eine andere Möglichkeit besteht darin, daß zwischen den Quellenprogrammen Absprachen über die benutzten Indexzellen getroffen werden. Gemäß der Absprache müssen dann durch den Pseudobefehl INDEX die entsprechenden Indexzellen fest zugeordnet oder es muß mit absoluten Indexadressen gearbeitet werden.

C

7. LITERALE

Literale ermöglichen eine zusätzliche Form der Adressierung. In bestimmten Fällen läßt sich mit Literalen Schreibarbeit einsparen und die Übersicht eines Programms erhöhen.

Ein Literal ist eine in Klammern eingeschlossene Folge von Informationseinheiten. Das Literal ist im Gegensatz zu Befehlen und Konstanten keine selbständige Informationseinheit, sondern es kann nur im Adressenteil eines TR 440-Befehls anstelle einer 16-Bit-Adresse (n, m oder z) auftreten. Diese Befehle dürfen dann nicht mit einer Spezifikation versehen werden. Literale in Literalen (sogenannte Literale 2. Ordnung) sind erlaubt, eine weitere Verschachtelung ist nicht zulässig. Ein Literal 1. Ordnung ist ein Literal, das nicht in einem anderen Literal enthalten ist. Bild 7.1 zeigt die allgemeine Form eines Literals und die sonst übliche Schreibweise eines Befehls mit einer Adresse. Die Wirkung ist in beiden Fällen gleich.

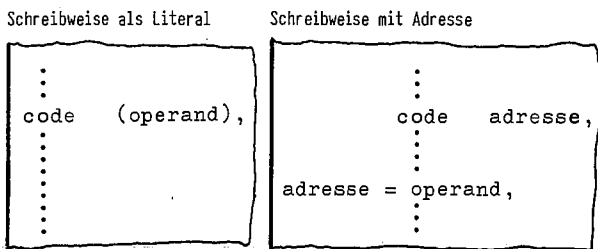


Bild 7.1 Allgemeine Form eines Literals

Informationseinheiten, die in Literalen enthalten sind, werden zwar wie normale Informationseinheiten übersetzt, aber nicht wie normale Informationseinheiten im Adressenraum angeordnet. Alle Informationseinheiten, die in Literalen 1. Ordnung enthalten sind, werden bei der Übersetzung in speziellen Ablagezonen (entsprechend dem impliziten Literalschlüssel 1) angeordnet. Die in Literalen 2. Ordnung enthaltenen Informationseinheiten werden ebenfalls in speziellen Ablagezonen (entsprechend dem impliziten Literalschlüssel 2) angeordnet.

Darüber hinaus kann mit Hilfe der im Abschnitt "Aufbau" beschriebenen Pseudobefehle, die die Ablage der Informationseinheiten steuern, vom Programmierer eine vom Normalfall abweichende Ablage der in Literalen enthaltenen Informationseinheiten erreicht werden.

Da Befehle innerhalb eines Literals im Speicher in andere Ablagezonen als die selbständigen Befehle gelegt werden, muß am Ende der Befehlsfolge im Literal ein Sprungbefehl stehen. Dieser Sprungbefehl soll verhindern, daß das gesamte Programm nach dem Durchlauf der Befehlsfolge im Literal beendet wird. Er muß also mit seiner Adresse auf die Stelle des Programms zielen, bei der der Programmablauf fortgesetzt werden soll.

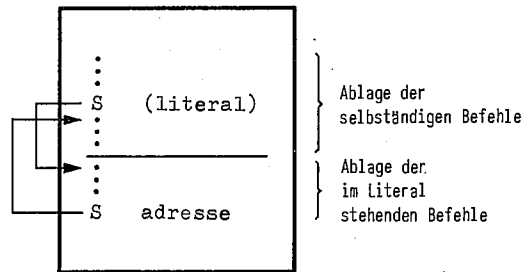


Bild 7.2 Beispiel für die Ablage der Befehle

7.1. Konstanten und Befehle in Literalen

Als Literal können eine Konstante oder ein Befehl bzw. Folgen von Konstanten und Befehlen geschrieben werden. Beispiele hierfür sind in den Bildern 7.3 bis 7.8 dargestellt. In der Schreibweise unterscheiden sich die innerhalb des Literals stehenden Konstanten und Befehle nicht von den selbständigen Konstanten und Befehlen. Auch in Literalen gelten die sonst für Konstanten üblichen Spezifikationen. Die in einem Literal stehenden Konstanten und Befehle dürfen jeweils einzeln benannt werden.

Enthält ein Literal als einzige Informationseinheit eine Konstante, die in ein Ganzwort übersetzt wird und in einem schreibgeschützten Teil des Adressenraums abgelegt werden soll, so wird diese Konstante nicht abgelegt, wenn bereits eine gleichartige Konstante abgelegt wurde. Als gleichartig werden Konstanten betrachtet, die in identische Bitfolgen übersetzt werden, die gleiche Typenbezeichnung aufweisen und im gleichen Teil des Adressenraums abgelegt werden sollen. Diese Konstanten werden Ganzwortliterals genannt. Ganzwortliterals werden unabhängig davon, ob es sich um Literale 1. oder 2. Ordnung handelt, in einer speziellen Ablagezone (entsprechend dem impliziten Literalschlüssel G) angeordnet.

7.2. Literale bei Bringebefehlen

Bei allen Befehlen, die Informationen aus dem Kernspeicher in Register bringen, kann der Operand anstelle seiner Kernspeicheradresse direkt angegeben werden.

Schreibweise als Literal

```

B (65536),
B (ANF=4096),
    
```

Schreibweise mit Adresse

```

B FAKT,
FAKT= 65536,
B ANF,
ANF= 4096,
    
```

Bild 7.3 Beispiele für Literale bei Bringebefehlen

Stehen in einem Literal bei einem Bringebefehl mehrere Konstanten, so muß durch Modifizierung dafür gesorgt werden, daß die auf die erste Konstante folgenden Konstanten angesprochen werden. Als Beispiel hierzu zeigt das Bild 7.4 einen Programmausschnitt, in dem durch Modifizierung und eine Schleife die Konstanten nacheinander gebracht werden. Wird nicht modifiziert und keine Schleife organisiert, dann wird nur die erste Konstante aus dem Literal gebracht.

```

:
:
ANF= ZX 6 X1,
      HXP -2 X1,
      M X1,
      B (1,16,256),
      SXN ANF,
:
:
    
```

Bild 7.4 Literal mit mehreren Konstanten

7.3. Literale bei Adressenbefehlen

Bei Befehlen BA, BAR, MA und XBA kann statt einer Adresse ein Versorgungsblock direkt folgen.

Im Bild 7.5 ist als Beispiel ein Versorgungsblock für ein Unterprogramm mit 4 Parametern dargestellt. Hierbei sollen im Unterprogramm von 100 Zahlen die Differenzen gebildet werden. Die Anfangsadressen der Zahlen sind symbolisch benannt worden.

Schreibweise als Literal

```

:
:
BA (MINU/A,
SUBTRA/A,
200/H,
DIFF/A),
:
SU UPR,
TR= B ANTON,
:
:
:
:
    
```

Schreibweise mit Adresse

```

:
:
BA VSB,
SU UPR,
TR= B ANTON,
:
:
VSB= MINU/A,
SUBTRA/A,
200/H,
DIFF/A,
:
:
    
```

Bild 7.5 Beispiel für Literal bei Adressenbefehl

7.4. Literale bei Sprungbefehlen

Bei bedingten oder unbedingten Sprungbefehlen kann anstelle der Sprungadresse eine kurze Befehlsfolge oder ein kleines Unterprogramm stehen.

Bei dem im Bild 7.7 gezeigten Programmausschnitt sollen Zahlen nach dem Vorzeichen sortiert und jeweils nach positiven und negativen Zahlen gesondert addiert werden. Eine Übersicht über den Programmablauf vermittelt das Flußdiagramm in Bild 7.6. Das Sortieren geschieht durch den Befehl SK0. Bei der Schreibweise als Literal steht im Adressenteil dieses Befehls ein Teilprogramm. Da die Befehle innerhalb eines Literals im Speicher an das Ende der selbständigen Befehle gelegt werden, muß am Ende der Befehlsfolge im Literal ein Sprungbefehl stehen, der auf einen nachfolgenden Befehl des Programms zielt. Wenn kein Sprungbefehl am Ende der Befehlsfolge im Literal steht, dann wird das gesamte Programm an dieser Stelle beendet. Im Programmablauf liegen die Befehlsfolgen, durch die die positiven und negativen Zahlen getrennt addiert werden, hintereinander. Darum muß bei jedem Schleifendurchlauf ein Teilprogramm übersprungen werden. Dies ist im Beispiel durch die eingezeichneten Pfeile hervorgehoben.

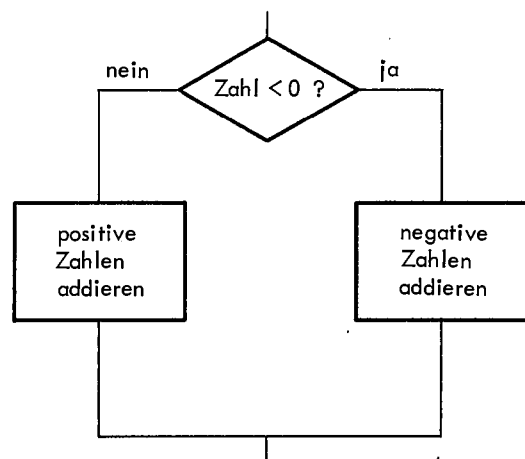
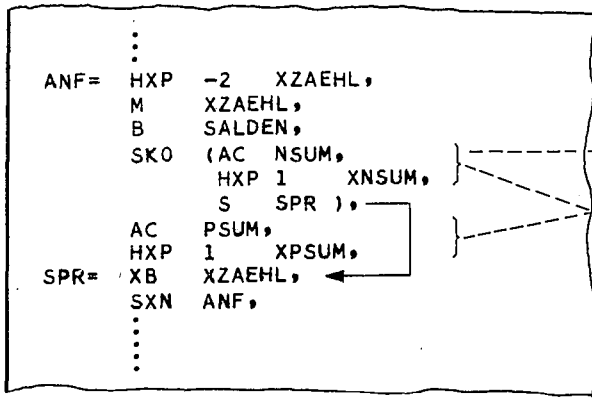


Bild 7.6 Flußdiagramm zum Programmbeispiel

Schreibweise als Literal



Schreibweise mit Adresse

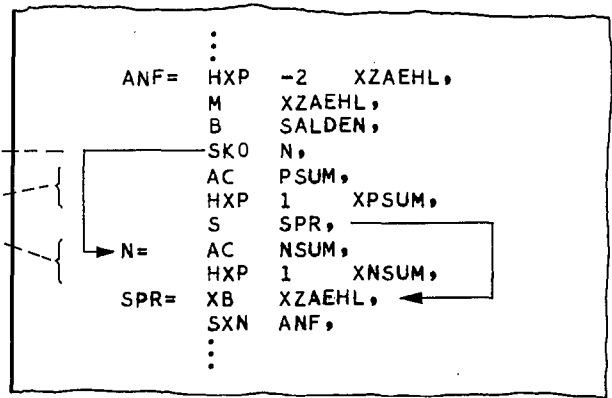


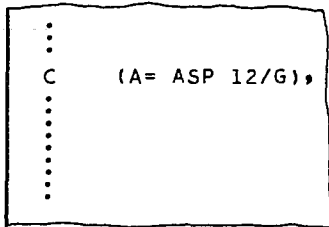
Bild 7.7 Beispiel für Literal bei Sprungbefehl

7.5. Literale bei Speicherbefehlen

Bei Speicherbefehlen kann statt der Adresse einer Freihalteanweisung diese direkt als Literal geschrieben werden.

Im Adressenteil eines Bringebefehls steht eine Literal-Befehlsfolge. Da Befehle in der zweiten oder einer der folgenden Großseiten abgelegt werden können, sind sie bei einem Bringebefehl mit einer 16-Bit-Adresse nicht adressierbar.

Schreibweise als Literal



Schreibweise mit Adresse

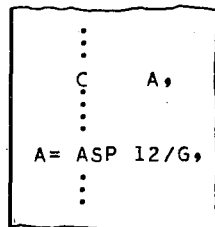


Bild 7.8 Beispiel für Arbeitsspeicherreservierung

Im Adressenteil eines bedingten oder unbedingten Sprungbefehls beginnt das Literal mit einer Konstanten. Sprünge können nur innerhalb der Befehls-Zone ausgeführt werden. Dadurch sind für Sprungbefehle die Konstanten in der Konstanten-Zone nicht erreichbar. Konstanten können aber auch in der Befehls-Zone abgelegt werden, wenn sie mit einer entsprechenden Spezifikation versehen sind.

7.6. Unzulässige Anwendung von Literalen

Eine unzulässige Benutzung eines Literals wird mit einer Fehlermeldung angezeigt, so zum Beispiel in den nachstehend aufgeführten Fällen:

Im Adressenteil eines Speicherbefehls steht eine Literal-Konstante. Konstanten werden in der schreibgeschützten Konstanten-Zone abgelegt, während der Speicherbefehl in die Variablen-Zone speichert.

C

TAS - Handbuch

Nov. 68

TELEFUNKEN
COMPUTER

D

TAS - HANDBUCH

Programmteile

INHALT

PROGRAMMTEILE



<u>1.</u>	<u>PRINZIP DER UNTERPROGRAMME</u>	1 - 1
1.1.	Aufruf und Rückkehr	1 - 1
1.2.	Unterprogrammstufen	1 - 1
1.3.	Parameterversorgung	1 - 2
1.4.	Speicherbedarf	1 - 3
1.5.	Indexspeicherbedarf	1 - 3
<u>2.</u>	<u>UNTERPROGRAMMAUFRUF ÜBER SFB UND SFBE</u>	2 - 1
2.1	Versorgungsparameter im Register	2 - 3
2.2.	Adresse des Versorgungsblocks im Register	2 - 3
2.3.	Adresse des Versorgungsblocks hinter SFB	2 - 8
2.4.	Versorgungsblock hinter SFB	2 - 8
2.5.	Versorgungsbefehle	2 - 9
2.6.	Versorgungsblock in Indexzellen	2 - 9
2.7.	Umschalten Indexbasis	2 - 10
<u>3.</u>	<u>UNTERPROGRAMMAUFRUF ÜBER SU ODER SUE</u>	3 - 1
3.1.	Versorgungsparameter im Register	3 - 6
3.2.	Versorgungsblock hinter SU	3 - 6
3.3.	Versorgungsbefehle	3 - 10
3.4.	Adresse des Versorgungsblocks hinter SU	3 - 11
3.5.	Adresse der Versorgungsblocks im Register	3 - 12
3.6.	Versorgungsblock in Indexzellen	3 - 13
3.7.	Umschalten der Indexbasis	3 - 14
<u>4.</u>	<u>ERSETZUNGSTECHNIK</u>	4 - 1
4.1.	Ersetzungsdefinition	4 - 1
4.2.	Ersetzungsbezug	4 - 1
4.3.	Einfache Ersetzung	4 - 1
4.4.	Verschachtelte Ersetzung	4 - 2
4.5.	Ersetzungsparameter	4 - 2
<u>7.</u>	<u>Makrotechnik</u>	7 - 1
7.1.	Elemente	7 - 1
7.1.1.	Anwendung	7 - 1
7.1.2.	Vergleich	7 - 3
7.2.	Einfache Makros	7 - 3
7.3.	Makros mit einem Parameter	7 - 4
7.4.	Makros mit mehreren Parametern (Param.-List.)	7 - 5
7.5.	Wiederholungen	7 - 8
7.5.1.	WIED-Befehl	7 - 9
7.5.2.	Wiederholungstext	7 - 9
7.5.3.	WEND-Befehl	7 - 9
7.6.	Versionen	7 - 10
7.6.1.	VERS-Befehl	7 - 11
7.6.2.	SØNST-Befehl	7 - 11
7.6.3.	VEND-Befehl	7 - 11
7.6.4.	Beispiele und Erläuterungen	7 - 12
7.7.	Globale Makrovariable	7 - 14
7.7.1.	VERSION*	7 - 14
7.7.2.	NUMMER*	7 - 14
7.7.3.	DATUM* und GENV*	7 - 17
7.8.	Beispiel mit Anwendung von Makros	7 - 18



D

8.	PROZEDUREN FÜR FORTRAN	8 - 1
8.1.	Aufruf einer FORTRAN-Prozedur	8 - 1
8.2.	Rücksprung in die aufgerufene Programmeinheit ..	8 - 5
8.3.	Aufbau der Prozedur	8 - 5
8.4.	Speicherbedarf	8 - 6
8.5.	Indexzellen	8 - 6
8.6.	Fehlerbehandlung	8 - 8
9.	PROZEDUREN FÜR ALGOL	9 - 1
9.1.	Verwendung der Indexspeicher	9 - 1
9.2.	Dienste der Freispeicherverwaltung	9 - 1
9.3.	Aufbau des Versorgungsblocks	9 - 2
9.4.	Aufruf einer ALGOL-Prozedur	9 - 2
9.5.	Kopf einer Codeprozedur nach ALGOL-Konventionen	9 - 2
9.6.	Zugriff auf aktuelle Parameter und lokale Größen	9 - 3
9.7.	Ausgang aus einer Codeprozedur	9 - 3
9.8.	Strukturerklärung	9 - 3
9.9.	Fehlerbehandlung	9 - 3
9.10.	Leistung der Versorgungsunterprogramme	9 - 4
9.11.	Aufbau der Zusatzversorgung	9 - 5
9.12.	Informationsverkehr eines Feldes	9 - 5
9.13.	Bereitstellung von Haupt- und Zusatzversorgung ..	9 - 6
9.14.	Beispiele	9 - 7

10. DER STANDARDDRAHMEN FÜR TAS-PROGRAMME .. 10 - 1
=====

- 10.1. Überblick 10 - 1
- 10.2. Allgemeine Regeln für den Standarddrahmen 10 - 1
- 10.3. Das Makro R&RAHMEN 10 - 2
- 10.4. Makros für Ein- und Ausgabe 10 - 2
- 10.5. Makros zum Beenden der Programmausführung 10 - 6
- 10.6. Beispiel 10 - 7

11. MAKROBIBLIOTHEKS-ORGANISATION 11 - 1
=====

- 11.1. Organisation einer Bibliothek als Datei 11 - 1
- 11.2. Der Begriff "Verfügbarkeit eines Makros" 11 - 1
- 11.3. Aufruf eines Makros 11 - 1
- 11.4. Pseudobefehle zum Beginnen und Beenden der Verfügbarkeit von Makros 11 - 1
- 11.5. Kommandos für Manipulationen mit Makro-Bibliotheken 11 - 2

1. PRINZIP DER UNTERPROGRAMME

Für die Beschreibung dieses Abschnittes wird davon ausgegangen, daß der Abschnitt "Adressenrechnung" bekannt ist. Die dort unter den Ersetzbefehlen beschriebenen Befehle MU und EMU werden hier im Zusammenhang nochmals erläutert.

Im Folgenden wird zuerst auf das Prinzip des Unterprogrammanschlusses eingegangen. Danach werden die beiden wichtigen Möglichkeiten besprochen, wie Unterprogramme aufgerufen werden und wie für die beiden Fälle das Unterprogramm organisiert sein muß. Schließlich wird ein Vergleich zwischen den beiden Möglichkeiten angestellt. Auf Grund der vielen möglichen Variationen, die durch den umfangreichen Befehlscode gegeben sind, erfolgt nur eine Gegenüberstellung einzelner Fakten, nicht aber eine globale Gegenüberstellung.

In weiteren Abschnitten werden die Konventionen für Standardunterprogramme behandelt.

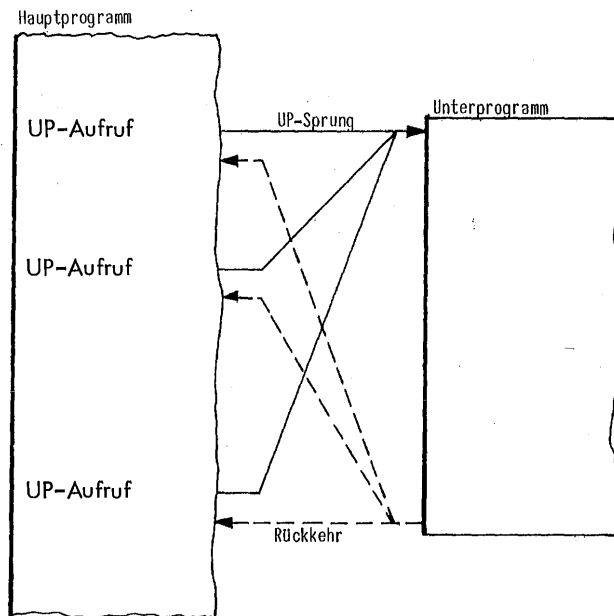


Bild 1.1 Prinzip des Unterprogrammaufrufs

1.1. Aufruf und Rückkehr

Ein Unterprogramm hat stets einen Namen und wird unter diesem Namen aufgerufen. Der Aufruf kann durch jeden Sprungbefehl erfolgen, der im Adressenteil eine Halbwordadresse hat. Es muß nur dafür gesorgt werden, daß das Unterprogramm die Adresse übermittelt bekommt, auf die es nach Erledigung seiner Aufgabe zurückkehren kann. Dies kann z.B. über die Versorgungsparameter geschehen (siehe dort).

Für den Aufruf von Unterprogrammen (Unterprogramm-sprung) sind drei spezielle Sprungbefehle vorgesehen. Es sind absolute Sprungbefehle, und sie machen dem Unterprogramm für den Rücksprung die Adresse des auf den Unterprogramm-sprung folgenden Befehls auf einfache Weise zugänglich, so daß daraus eine vereinfachte Programmierung resultiert. Es sind dies einmal der Befehl SFB und zum anderen die Befehle SU und SUE. Diese beiden Befehlsgruppen bedingen einen verschiedenen Aufbau der Unterprogramme. Sie sind daher in den Abschnitten 2. und 3. getrennt behandelt.

Aus dem Vorhergesagten resultiert, daß ein Unterprogramm über einen Sprungbefehl aufgerufen wird. Im Adressenteil des Sprungbefehls steht der Name des Unterprogramms. Die Befehle SFB, SU und SUE sind besonders für den Unterprogrammanschluß geschaffen. Es kann an jede beliebige Stelle im Hauptprogramm zurückgekehrt werden. In den meisten Fällen wird man jedoch auf den Befehl zurückspringen, der nach dem Befehl für den Sprung ins Unterprogramm steht.

Bei umfangreichen Programmen kann es vorkommen, daß das Unterprogramm nicht mehr in der Großseite 1 untergebracht werden kann und dann bei der Montage in eine andere Großseite gelegt wird. Für diesen Fall muß darauf geachtet werden, daß Unterprogramm-sprung und Rückkehr mit einem Sprungbefehl erfolgen, der in eine andere Großseite springen kann. Es sind dies die Sprungbefehle SFBE, SUE, SE und die mit MABI und MU modifizierten Sprungbefehle, die ein Halbword adressieren.

1.2. Unterprogrammstufen

Es ist möglich, von einem Unterprogramm in ein weiteres zu springen und von dort wieder in ein weiteres. Diese Unterprogramm-kette ist nicht begrenzt. Betrachten wir jeweils das Programm, in dem der Sprung in das Unterprogramm steht, wiederum als Hauptprogramm, so gilt das in Abschnitt 1.1. Gesagte.

Geht man in der Betrachtungsweise von einem der Unterprogramme in der Unterprogramm-kette aus, so ist das Programm, von dem aus in dieses Programm gesprungen wurde, das "übergeordnete Programm" und das Programm, in das gesprungen wird, das "untergeordnete Programm". Bei Aufruf eines Unterprogramms wird also in das untergeordnete Programm gesprungen.

D

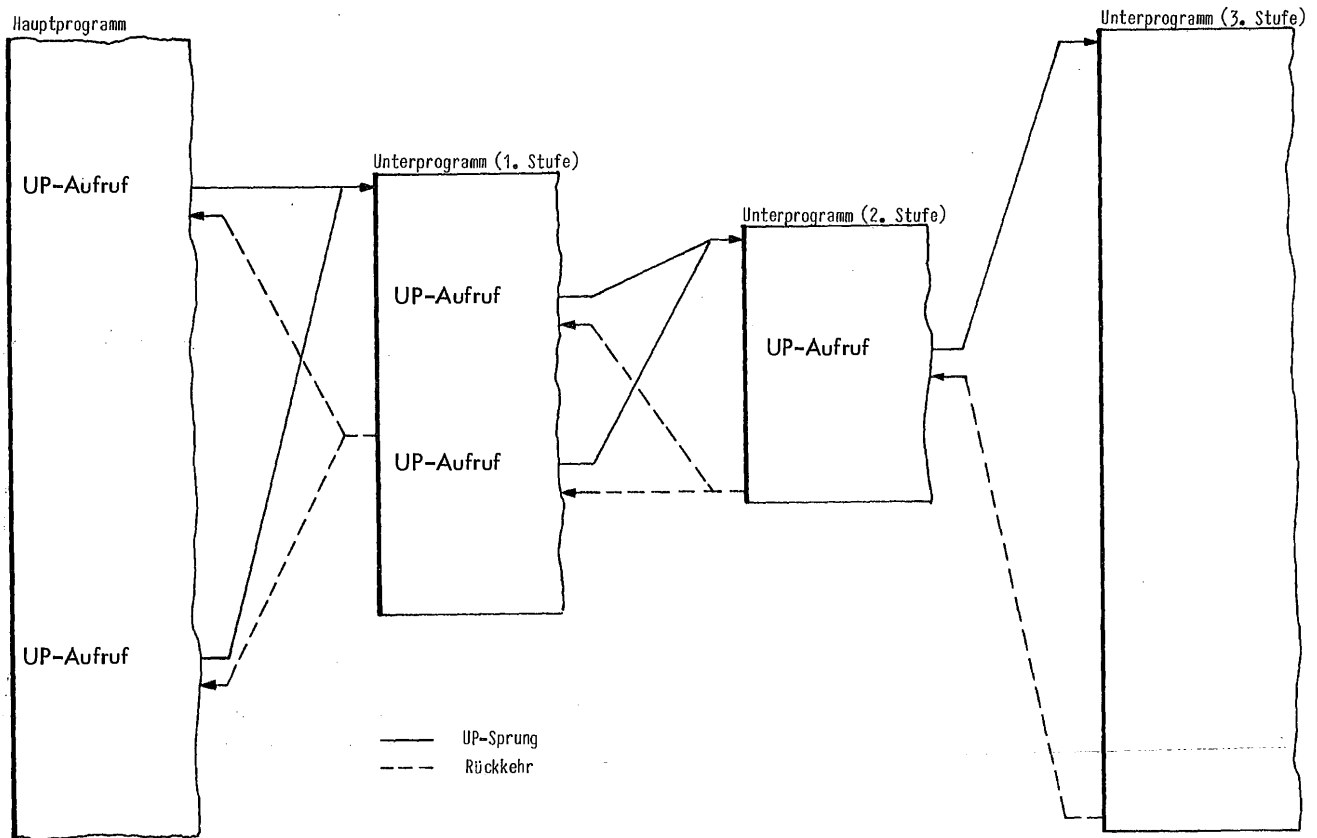


Bild 1.2 Mehrstufiger Unterprogrammaufruf

Die Rückkehr aus der Unterprogrammreihe erfolgt schrittweise in der umgekehrten Reihenfolge wie der Aufruf der Unterprogramme, d.h. ein Unterprogramm springt immer in das übergeordnete Programm zurück. Wie dies auch anders gemacht werden kann, soll hier nicht erwähnt werden.

Das erste Unterprogramm in der Unterprogrammreihe, also das Unterprogramm, in das vom Hauptprogramm gesprungen wird, ist die erste Unterprogrammstufe. Das zweite Unterprogramm, in das von der ersten Stufe gesprungen wird, ist die zweite Stufe, usw. Es können x Unterprogrammstufen vorhanden sein. Diese Unterprogrammstufen sind interessant für den Fall, daß das Unterprogramm mit dem Befehl SU oder SUE aufgerufen wird.

1.3. Parameterversorgung

Die Tätigkeit des Unterprogramms ist durch seine Befehlsfolge festgelegt. Beim Aufruf des Unterprogramms müssen ihm noch die zu verarbeitenden Daten zugänglich gemacht werden. Dies wird je nach Aufgabenstellung auf verschiedene Art gemacht.

Benötigt das Unterprogramm nur wenige Daten, so können sie in den verschiedenen Registern untergebracht werden. Durch den Aufruf der Unterprogramme werden lediglich die Inhalte folgender Register verändert:

Befehlszähler	F
Bereitschaftsregister	B (nur bei SFB)
Unterprogrammregister	U (nur bei SU und SUE)

Die anderen Register bleiben unverändert, so daß das Unterprogramm die Inhalte dieser Register weiterverarbeiten kann. Fallen als Ergebnis ebenfalls nur wenige Daten an, so können sie in den Registern hinterlassen werden, wobei das Register F und - abhängig davon, mit welcher Befehlskombination zurückgesprungen wird - ggf. die Register B und U nicht benutzt werden können. Als Beispiel sei genannt, daß ein Unterprogramm die dritte Wurzel aus einer ganzen Zahl zieht. Bevor das Unterprogramm aufgerufen wird, kann die Zahl, aus der die Wurzel gezogen werden soll, im Register A bereitgestellt werden. Das Unterprogramm kann dann das Ergebnis wieder im Register A hinterlassen.

Eine weitere Möglichkeit ist die Angabe eines Versorgungsblocks. Dieser Versorgungsblock enthält alle Angaben, die das Unterprogramm benötigt. Der Versorgungsblock kann mehrere Halb- oder Ganzwörter umfassen. Wie der Versorgungsblock aufgebaut sein muß, liegt durch den Aufbau des Unterprogramms fest. Es gibt eine Reihe von Möglichkeiten, von denen die wichtigsten hier erläutert werden.

- Der Versorgungsblock kann an einer beliebigen Stelle stehen. Die Anfangsadresse des Versorgungsblocks kann in eines der Register gebracht werden.

- Der Versorgungsblock kann unmittelbar hinter dem Befehl stehen, mit dem in das Unterprogramm gesprungen wurde. Der Rücksprung erfolgt auf den ersten Befehl hinter dem Versorgungsblock. Da Befehle schreibgeschützt sind, kann dieser Versorgungsblock nur Konstanten enthalten.
- Der Versorgungsblock kann an beliebiger Stelle stehen. Die Anfangsadresse des Versorgungsblocks steht hinter dem Befehl, mit dem im Unterprogramm gesprungen wurde. Der Rücksprung erfolgt auf den ersten Befehl hinter der Anfangsadresse.

Der Versorgungsblock kann beliebig lang sein und einen beliebigen Aufbau haben. Er muß jedoch den Konventionen genügen, die durch den Aufbau der Unterprogramme festgelegt wurden. Der Versorgungsblock kann folgende Angaben enthalten:

- die Operanden selbst
- die Adresse von Operanden
- die Anfangsadresse einer Gruppe von Operanden
- die Länge eines Bereiches
- die Anzahl der Wörter, die verarbeitet werden sollen
- Angabe über Abbruchkriterien
- Rücksprungadresse für den Normalfall
- Rücksprungadresse für Sonderfälle
- einen Befehl, der im Unterprogramm über den Befehl T zur Ausführung kommt (Versorgungsbefehle).

Besonders beachtet werden muß noch, daß alles, was innerhalb einer Befehlsfolge (im B-Bereich) steht, ebenso wie die Befehle, schreibgeschützt ist, also nicht vom Programm her verändert werden kann. So ist es z.B. nicht sinnvoll, die Operanden selbst hinter den Sprungbefehl zu legen; das gleiche trifft für die Ergebnisse zu.

Die vorstehend angegebenen Möglichkeiten können, soweit sinnvoll, kombiniert werden. Soll z.B. eine Anzahl von Zahlen aufsummiert werden, so können Anfangsadresse und Anzahl der Zahlen in einem Versorgungsblock angegeben und das Ergebnis dem Register übergeben werden.

1.4. Speicherbedarf

Ebenso wie jedes andere Programm hat auch ein Unterprogramm einen Speicherbedarf. Dieser Bedarf betrifft zumindest den B-Bereich, in dem die Befehlsfolge liegt. Des weiteren kann auch Bedarf in anderen Bereichen (K, V, D) auftreten. Ferner kann das Unterprogramm Indexspeicherzellen benötigen.

Im allgemeinen wird man versuchen, ein Unterprogramm so selbständig zu machen, daß sein Speicherbedarf im Hauptprogramm bzw. im übergeordneten Programm nicht berücksichtigt zu werden braucht. Das läßt sich auch weitgehend erreichen.

Man muß zwischen dem Speicherbedarf unterscheiden, der nur vom Unterprogramm benötigt wird, und dem, der vom Unterprogramm und übergeordneten Programm gemeinsam benutzt wird. Daten, die vom Unterprogramm verarbeitet werden sollen, werden vom übergeordneten Programm in einem Speicherbereich bereitgestellt, der diesem Programm gehört und auch dort deklariert wird. Das Unterprogramm benutzt ihn lediglich mit; er gehört also nicht zum Unterprogramm. Das gleiche gilt für den Speicherbereich, in dem die Ergebnisse des Unterprogramms vom Unterprogramm abgelegt werden.

Zum anderen kann aber auch das Unterprogramm für Zwischenergebnisse und Konstanten, von denen das Hauptprogramm nichts weiß und auch nichts zu wissen braucht, Speicherraum benötigen. Diesen Speicherraum wird es jedoch auch selbst festlegen und deklarieren, so daß er im Hauptprogramm nicht berücksichtigt zu werden braucht.

1.5. Indexspeicherbedarf

Beim Assemblieren eines Programms oder Teilprogramms wird alles, was zwischen dem ersten Pseudobefehl SEGM und dem Pseudobefehl ENDE steht, in bezug auf Indexadressen als Einheit betrachtet. Allen Indexadressen werden in der Reihenfolge, wie sie dem Assembler übergeben werden, absolute Indexadressen zugeordnet. Der ersten symbolischen Indexadresse wird die absolute Adresse 0 zugeordnet.

Mit dem Pseudobefehl INDEX ist es möglich, eine Zuordnung von Indexzellen zu erreichen, die von der oben beschriebenen Zuordnung abweicht. Weitere Einzelheiten dazu sind dem Abschnitt "Befehle und Adressierung" zu entnehmen.

In bezug auf die Verwendung von Indexzellen in Unterprogrammen müssen wir unterscheiden, ob das Unterprogramm gleichzeitig mit dem Hauptprogramm assembliert wird oder ob das Unterprogramm als Teilprogramm für sich assembliert und später durch das Montageprogramm an das Hauptprogramm angeschlossen wird. Im letzten Fall gibt es mehrere Möglichkeiten.

- o Das Hauptprogramm reserviert für das Unterprogramm die benötigten Indexzellen.
- o Das Unterprogramm definiert einen neuen Indexbereich.

Die Möglichkeiten können auch miteinander sinnvoll kombiniert werden.

Bei der Beschreibung des Unterprogramms wäre es zweckmäßig anzugeben, welche Indexzellen verwendet werden. Bei diesen Angaben müssen auch die Indexzellen enthalten sein, die die von diesem Unterprogramm verwendeten weiteren Unterprogramme benötigen.

D

TAS - Handbuch

Dez. 68

1.5.1. Unterprogramm wird mit Hauptprogramm assembliert

Werden Haupt- und Unterprogramm zusammen assembliert und nicht in Teilprogramme aufgeteilt, so brauchen keine besonderen Vorkehrungen getroffen zu werden. Das Vorstehende gilt auch, wenn das Unterprogramm zusammen mit einem Teil des Hauptprogramms als Teilprogramm assembliert wird. Wie die Indexadressen bei der Montage der Teilprogramme behandelt werden, soll hier nicht erläutert werden. Näheres dazu ist dem Abschnitt "Befehle und Adressierung" zu entnehmen.

1.5.2. Hauptprogramm reserviert Indexzellen

Das Unterprogramm kann auch beliebige Indexadressen verwenden, wenn im Hauptprogramm dafür gesorgt wird, daß diese Indexzellen für das Unterprogramm reserviert sind. Diese Reservierung kann mit dem Pseudobefehl INDEX erfolgen. Benötigt das Unterprogramm z.B. 10 Indexzellen und beginnen diese bei der Adresse 0, so müssen im Hauptprogramm mit dem Pseudobefehl

```
INDEX 0 (10),
```

diese 10 Indexzellen für das Unterprogramm (ggf. mit seinen weiteren Unterprogrammstufen) reserviert werden.

Im Unterprogramm können dann die absoluten Adressen 0 bis 9 verwendet werden. Werden symbolische Adressen verwendet, so werden ihnen die absoluten Adressen, beginnend bei 0, zugeordnet. Soll eine andere Zuordnung erfolgen, so ist dies mit dem Pseudobefehl INDEX vorzunehmen. In Bild 1.3 ist ein Beispiel gezeigt, bei dem die Indexzellen 3, 4, 5 und 6 belegt werden.

```

:
:
INDEX 3(XANZ,X1,ZAEHL,X2),
:
:

```

symbolische Adressen	absolute Adressen
XANZ	3
X1	4
ZAEHL	5
X2	6

Bild 1.3 Beispiel Zuordnung absoluter Indexadressen

1.5.3. Gemeinsame Benutzung von Indexzellen

Indexzellen, die gemeinsam vom Haupt- und Unterprogramm benutzt werden, sind von einem der beiden Programme festgelegt. Die absoluten Adressen sind identisch. Werden symbolische Adressen verwendet, so müssen die Namen nicht identisch sein; es ist jedoch dafür zu sorgen, daß die den Namen zugeordneten absoluten Adressen identisch sind.

Hauptprogramm

```

:
:
INDEX 0(16),
INDEX 20(ANF,NAME,K),
INDEX 25(3),
:
:

```

für Unterprogramm freigehalten

gemeinsam mit UP2

für UP2 freigehalten

Indexzellen 0 bis 15 zur freien Verwendung der Unterprogramme und des Hauptprogrammes; können nach Unterprogrammaufruf verändert sein.

Unterprogramm

```

UP1= SEGM,
XC X1,
:
TTX X2 X3,
ZX 3 X4,
:
ENDE,

```

getrennt assembliert, implizite Benennung
4 Indexzellen: 0, 1, 2, 3

Unterprogramm

```

UP2= SEGM,
INDEX 20(AN,N,K),
INDEX 25(ANF,NR,Z),
:
ZX 100 5,
TTX 6 7,
:

```

eigene Indexzellen: 5, 6, 7
gemeinsame Indexzellen: 20, 21, 22
vom Hauptprogramm freigehalten: 25, 26, 27

Bild 1.4 Indexspeicherbedarf der Unterprogramme

Dies geschieht dadurch, daß in beiden Programmen mit dem Pseudobefehl INDEX ein gemeinsamer Pegel festgelegt wird und die Indexnamen in der entsprechenden Reihenfolge aufgeführt werden. Dies sei an folgendem Beispiel gezeigt:

im Hauptprogramm: INDEX 10 (ANZ, X1, LAENG, X2),				
im Unterprogramm: INDEX 10 (A, X1, L, X2),				
absolute Adressen				
in beiden Programmen:	10,	11, 12,		13,

Das Setzen einer neuen Indexbasis für das Unterprogramm kann sowohl vom Unterprogramm als auch vom Hauptprogramm her erfolgen. Das Programm, das die Indexbasis neu setzt, muß dafür sorgen, daß ein entsprechender Speicherbereich mit dem Pseudobefehl ASP freigehalten wird, und zwar soviel Halbwörter, wie Indexzellen benötigt werden.

Es ist auch möglich, daß der neu definierte Indexbereich mitten im alten Indexbereich liegt. Das heißt, daß Indexzellen des Hauptprogramms gleichzeitig vom Unterprogramm mit benutzt werden, jedoch unter anderen Indexadressen (siehe dazu Bild 1.5).

Das Setzen einer neuen Indexbasis ist für den Unterprogrammaufruf mit dem Befehl SFB im Abschnitt 2.7. und mit den Befehlen SU und SUE im Abschnitt 3.7. beschrieben.

1.5.4. Setzen einer Indexbasis

Das Unterprogramm kann mit Hilfe des Befehls BCI oder ZI einen neuen Indexbereich für seinen eigenen Bedarf festlegen. Damit ist es in bezug auf seinen Indexbedarf unabhängig vom Hauptprogramm.

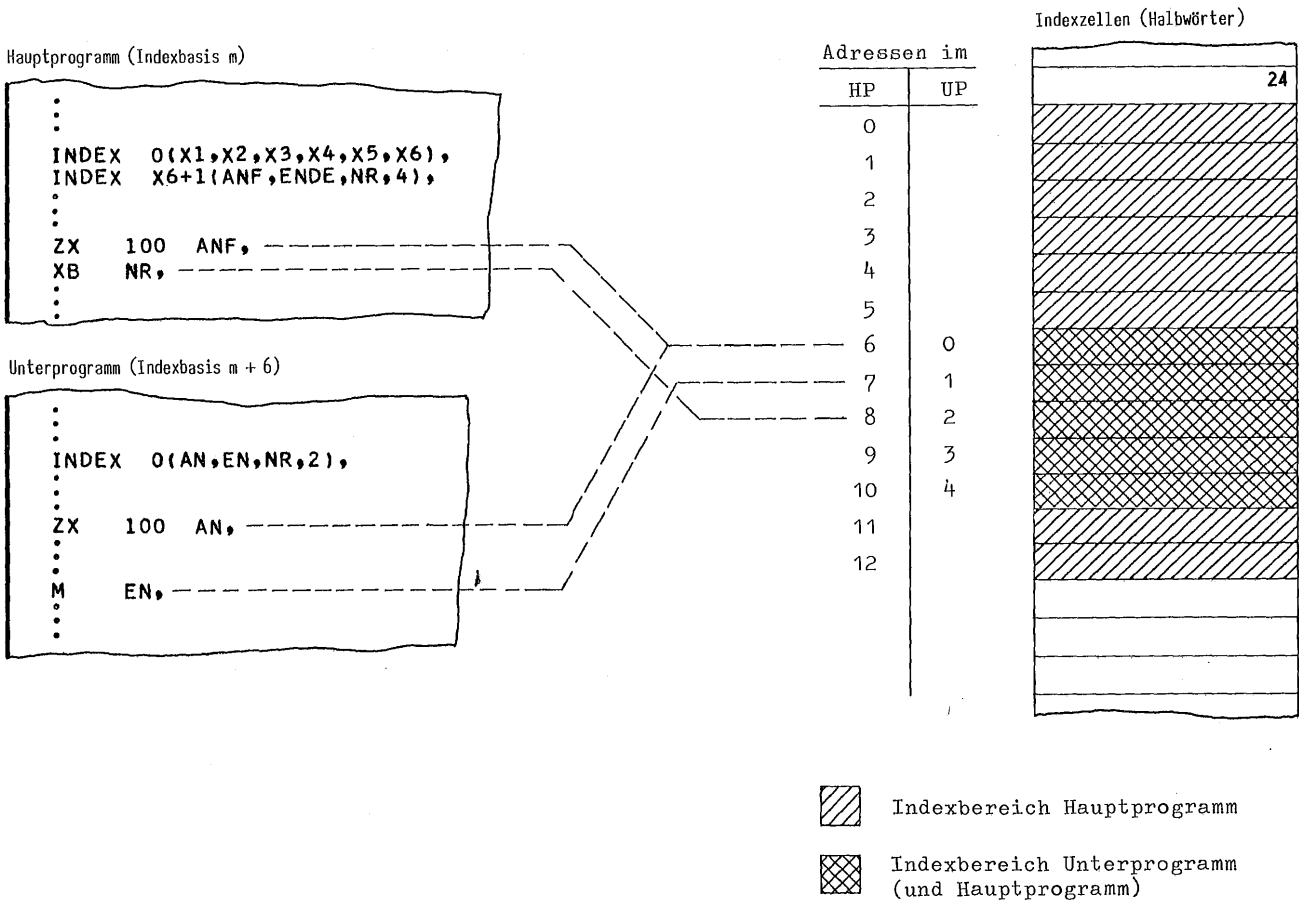


Bild 1.5 Überschneiden zweier Indexbasen

D

2. UNTERPROGRAMMAUFRUF ÜBER SFB UND SFBE

Für den Sprung in ein Unterprogramm stehen die Befehle SFB und SFBE zur Verfügung. Sie sind unbedingte Sprungbefehle.

SFB	m	Sprünge und bringe $\langle F \rangle + 1$ nach <u>B</u>	$\langle B \rangle := \langle F \rangle + 1$ $\langle F \rangle_{9-24} := m$
-----	---	--	---

Der Befehl SFB bringt die Adresse des auf ihn folgenden Befehls in das Register B. Anschließend wird das Register F auf den im Adressenteil des Befehls angegebenen Wert neu gesetzt. Dies ist die Adresse des Unterprogramms. Damit ist der Sprung in das Unterprogramm durchgeführt. Gesprungen werden kann nur innerhalb einer Großseite. Soll in eine andere Großseite gesprungen werden, so muß der Befehl SFB mit dem Befehl MABI modifiziert werden. Der Befehl kann auf 2. Art modifiziert werden.

Soll in ein Unterprogramm gesprungen werden, so steht im Adressenteil des Befehls der Name des Unterprogramms. Ist der Befehl ausgeführt, so wird im Unterprogramm fortgefahren. Im Register B steht die Adresse des auf den Befehl SFB folgenden Befehls. Wir wollen sie die "technische Rücksprungadresse" nennen. Im einfachen Fall wird bei der Rückkehr aus dem Unterprogramm in das Hauptprogramm bei dieser Adresse fortgefahren. Damit es am Ende seiner Aufgabe an diese Stelle zurückkehren kann, muß das Unterprogramm, bevor es das Register B verändert, diese technische Rücksprungadresse sicherstellen. Das kann geschehen, indem es den Inhalt des Registers B entweder mit dem Befehl XC in eine Indexzelle oder mit dem Befehl TBC in eine Speicherzelle (Halbwort) abspeichert.

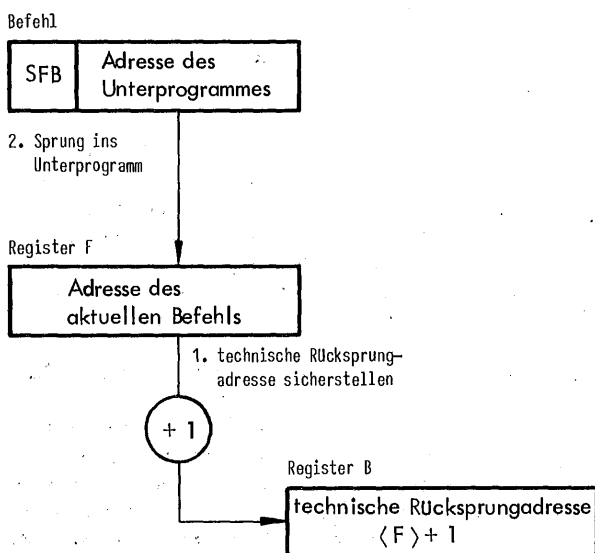


Bild 2.1a Unterprogrammierung mit Befehl SFB

Liegt ein Unterprogramm in einer anderen Großseite als das Hauptprogramm, so ist der Sprung ins Unterprogramm mit dem Befehl SFB nur möglich, wenn der Befehl SFB mit MABI modifiziert wird oder wenn statt dessen der Befehl

SFBE	m	Sprünge und bringe $\langle F \rangle + 1$ nach <u>B</u> nach <u>Ersetzung</u>	$\langle B \rangle := \langle F \rangle + 1$ $\langle F \rangle := \langle m \rangle$
------	---	--	--

angewendet wird. Er hat die gleiche Wirkung wie der Befehl SFB, nur steht im Adressenteil des Befehls SFBE statt der Adresse des Unterprogramms die Adresse eines Halbwortes, welches die Adresse des Unterprogramms enthält. Die Adresse kann dann eine 24-Bit-Größe sein, d.h. mit dem Befehl SFBE ist ein Unterprogramm erreichbar, das in einer anderen Großseite liegt. (Beim Befehl SFB ist der Sprung in eine andere Großseite nur möglich, wenn der Befehl SFB mit dem Befehl MABI modifiziert wurde.)

Das in den folgenden Abschnitten für den Befehl SFB |Beschriebene gilt entsprechend auch für den Befehl SFBE.

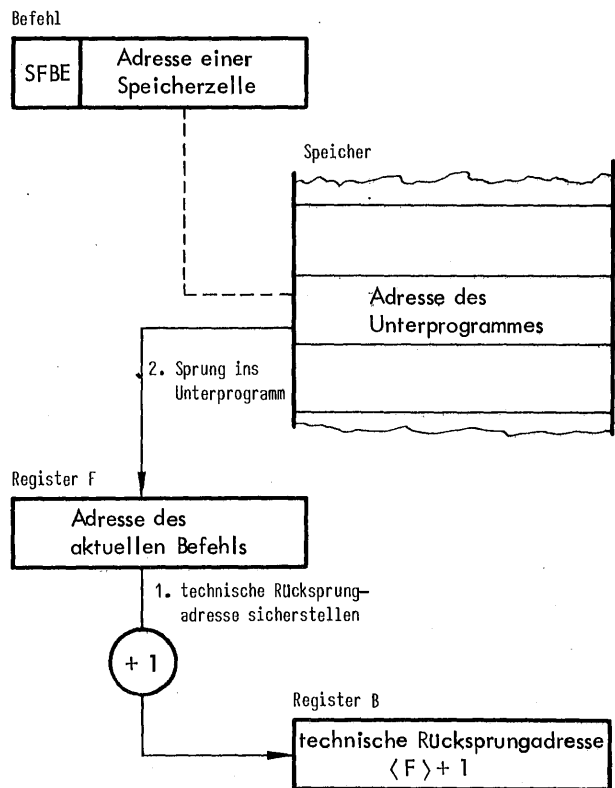


Bild 2.1b Unterprogrammierung mit Befehl SFBE

Hauptprogramm

```

SFB UP,
ADRUP=UP/A,
SFBE ADRUP,
SFBE (UP/A),
    
```

Bild 2.2 Beispiele für die Befehle SFB und SFBE

Die Rückkehr vom Unterprogramm ins Hauptprogramm kann mit jedem der Sprungbefehle erfolgen, die im Adressenteil ein Halbwort adressieren. Die Art des Rücksprungs hängt einmal davon ab, bei welcher Bedingung das Unterprogramm verlassen wird. Dies hat Einfluß auf den Sprungbefehl. Zum anderen hängt sie davon ab, ob dem Befehl SFB oder SFBE eine Versorgungsinformation folgt oder nicht. In den nachfolgenden Abschnitten wird darauf jeweils eingegangen.

Wurde die technische Rücksprungadresse in einer Indexzelle abgespeichert, so kann im einfachen Fall mit dem Befehl E auf die technische Rücksprungadresse zurückgekehrt werden. Mit Hilfe der Modifizierbefehle M, MF und MFU kann auf ein Halbwort gesprungen werden, das relativ zur technischen Rücksprungadresse liegt. In eine andere Großseite kann mit Hilfe des Befehls MABI gesprungen werden.

Wurde die technische Rücksprungadresse in einer Speicherzelle abgespeichert, so kann im einfachen Fall mit dem Befehl SE zurückgekehrt werden. Um auf ein Halbwort zu springen, das relativ zur technischen Rücksprungadresse steht, können die Befehle MCF, MCFU, MAB und MABI verwendet werden. Ein Sprung in eine andere Großseite ist mit dem Befehl SE und mit Hilfe von MABI möglich.

```

SE TRS,
MCF TRS,
SI 3,

MCFU TRS,
SXG 3,

MC TRS,
SKG 3,

TCB TRS,
MAB SNO 3,

TCB TRS,
MABI SXI 3,
    
```

{ Rücksprung in andere Großseite möglich

{ Rücksprung in andere Großseite möglich

technische Rücksprungadresse in der Speicherzelle TRS

Bild 2.3b Rücksprungbeispiele (TRS = Speicherzelle)

```

E SGG TRX,
M TRX,
SI 3,
-----
MFU TRX,
SXG 3,
MF TRX,
S 3,
-----
XB TRX,
MABI SNO 3,
    
```

{ Rücksprung auf technische Rücksprungadresse

{ Rücksprung auf technische Rücksprungadresse + 3

{ Rücksprung in andere Großseite möglich

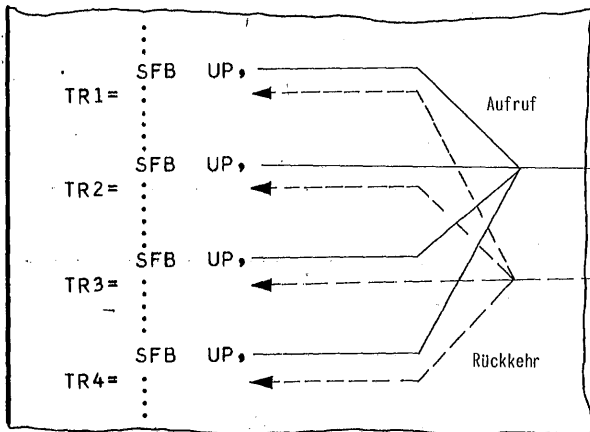
technische Rücksprungadresse in der Indexzelle TRX

Bild 2.3a Rücksprungbeispiele (TRX= Indexzelle)

Bild 2.4a zeigt in einem Beispiel den Aufruf eines Unterprogramms mit Hilfe des Befehls SFB und die Rückkehr in das Hauptprogramm auf die technische Rücksprungadresse. Das Bild 2.4b zeigt dieses Beispiel nach dem Assemblieren (mit absoluter Adresse), und im Bild 2.4c ist der Ablauf des Programms gezeigt. Die gestrichelte Linie deutet an, daß der Wert nicht verändert wird.

Die Versorgung des Unterprogramms mit den nötigen Daten geschieht über die Versorgungsparameter. Sie umfassen sowohl Angaben über die Eingangsdaten als auch über die Ausgangsdaten (vom Unterprogramm aus gesehen). Die Übergabe der Versorgungsparameter kann auf verschiedene Arten erfolgen. In den nachfolgenden Abschnitten sind die wichtigsten Möglichkeiten aufgezeigt. Die verschiedenen Möglichkeiten können auch - soweit sinnvoll - miteinander kombiniert werden.

Hauptprogramm



Unterprogramm

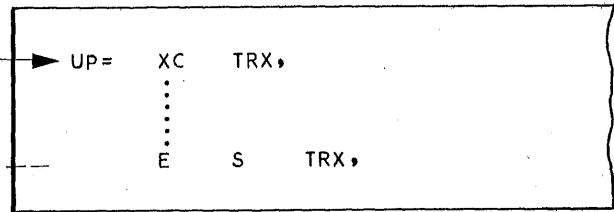
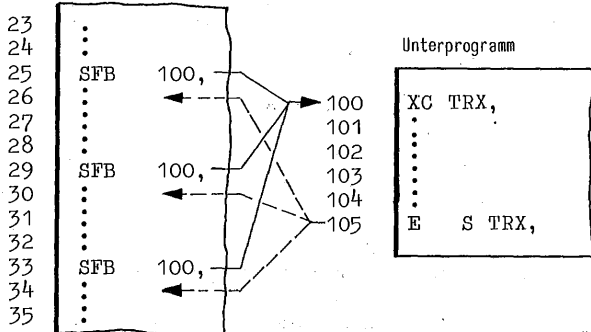


Bild 2.4a Beispiel Unterprogrammaufruf mit SFB

Hauptprogramm



Unterprogramm

2.1. Versorgungsparameter im Register

Benötigt das Unterprogramm nur einzelne Daten, so können diese dem Unterprogramm in den Registern übergeben werden. Gelegentlich sind die Daten noch von den vorhergehenden Operationen im Register oder sie werden durch einen oder mehrere Bringe-Befehle in die Register gebracht. Dies stellt die einfachste Übergabe der Versorgungsparameter dar.

Für die Ergebnisse, die das Unterprogramm liefert, gilt das gleiche. Sind es nur einzelne Werte, so können sie in den Registern hinterlassen und vom Hauptprogramm weiterverarbeitet oder abgespeichert werden.

Im Bild 2.5 wird die Summe $a + b$ dem Unterprogramm SINUS im Register A übergeben. Dieses liefert das Ergebnis, also den Sinus dieses Wertes, im Register A an das Hauptprogramm zurück. Der Wert wird zwischengespeichert und der Wert c in das Register A gebracht. Das Unterprogramm COSINUS liefert den Cosinuswert von c in das Register A. Hierzu wird der zwischengespeicherte Sinuswert von $a + b$ hinzuaddiert.

In den Bildern 2.6a bis 2.6b ist als Beispiel ein Unterprogramm zur Unterdrückung führender Nullen aufgeführt. Das Hauptprogramm stellt im Register A eine sechsstellige Dezimalzahl im Zentralcode zur Verfügung. Das Unterprogramm setzt für die führenden Nullen das Zeichen "Leertaste" ein (Oktade AF). Die Zahl 000347 wurde in die Form $\square\square\square 347$ umgewandelt ($\square =$ Leertaste). Das Ergebnis wird wiederum im Register A dem Hauptprogramm übergeben.

Register F	Befehl	Programm	Register B	Indezelle TRX
23	⋮			
24	⋮			
25	SFB 100,		26	↓
100	XC TRX	Unterprogramm		26
101	⋮			
102	⋮			
103	⋮			
104	⋮			
105	E S TRX			
26	⋮	Hauptprogramm		
27	⋮			
28	⋮			
29	SFB 100,		30	↓
100	XC TRX	Unterprogramm		30
101	⋮			
102	⋮			
103	⋮			
104	⋮			
105	E S TRX			
30	⋮	Hauptprogramm		
31	⋮			↓

Bild 2.4c Ablauf des Beispiels aus Bild 2.4a

2.2. Adresse des Versorgungsblocks im Register

In vielen Fällen ist es günstiger, die Versorgungsparameter für das Unterprogramm in einem Versorgungsblock zusammenzufassen. Sind mehrere Parameter zu übergeben, so bildet dies die einzige Möglichkeit.

Hauptprogramm

```

:
:
B   A,
A   B,
SFB SINUS,
TR1= C   SIN,
      B   C,
      SFB COSINUS,
TR2= AQ  SIN,
:
:

```

Unterprogramm

```

SINUS.=XC TRX,
:
:
:
E   S   TRX,

```

Unterprogramm

```

COSINUS.=XC TRX,
:
:
:
E   S   TRX,

```

$$y = \sin(a + b) + \cos c$$

Bild 2.5 Beispiel für Versorgungsparameter im Register

Hauptprogramm

```

:
SFB NULU,
:
:
SFB NULU,
:
:
SFB NULU,
:
:
SFB NULU,
:
:

```

Unterprogramm

```

NULU= XC TRX,
ZX 48 XZ,
BH ('0''/R),
RT AQ,
ANF= LR A3,
SH ZL 8,
HXP -8 XZ,
SXI SCHL,
SI ANF,
SCHL= A ('AFAFAFAF00'/3),
M XZ,
SH ZL 0,
E S TRX,

```

Unterdrückung führender Nullen

Bild 2.6a Beispiel für Versorgungsparameter im Register

Unterprogrammname:	NULU	Nullunterdrückung
Aufgabe:	Im Register A stehen 6 Ziffern im Zentralcode (0...9). Führende Nullen werden durch Leertasten ersetzt. Ergebnis steht wieder im Register A.	
Aufruf:	SFB NULU,	
Register:	⟨A⟩ = 6 Ziffern oktaden im Zentralcode (ZC1) Ziffern 0...9	
Rückkehr:	Adresse des Befehls SFB + 1	
Register:	⟨A⟩ = Zahl im Zentralcode (ZC1) links mit Leertasten aufgefüllt	
Speicherbedarf:	K-Bereich	= 2 Ganzwörter
	V-Bereich	= -
	B-Bereich	= 13 Halbwörter
	D-Bereich	= -
	Indezellen	= 2 (Adressen 0,1)

Bild 2.6c Angaben zum Unterprogramm aus Bild 2.6a

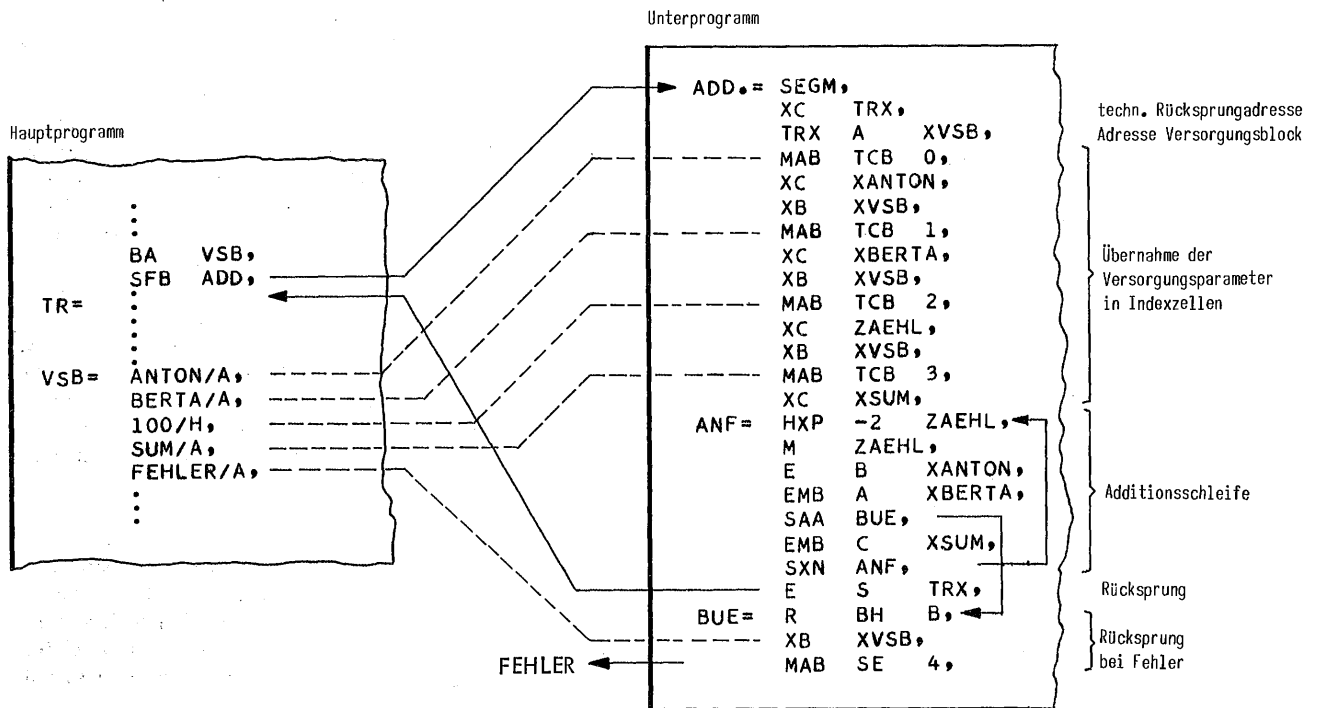


Bild 2.8a Beispiel für Adresse des Versorgungsblocks im Register

Unterprogrammname: ADD Addition

Aufgabe: $a_x + b_x = c_x$, $x = n-2, n-4, \dots, 6, 4, 2, 0$.

Aufruf: SFB ADD,

Register: <A> = Adresse des Versorgungsblocks

Versorgungsblock: a/A, a = Anfangsadresse erster Summanden
 b/A, b = Anfangsadresse zweiter Summanden
 n/H, n = Anzahl der Durchläufe mal 2
 c/A, c = Anfangsadresse der Summen
 f/A, f = Rücksprungadresse bei Bereichsüberschreitung

Rückkehr: Adresse des Befehls SFB + 1

Fehlerausgang: wenn $c_x \geq 2^{46}$: Rücksprung nach f

Speicherbedarf: K-Bereich = --
 V-Bereich = --
 B-Bereich = 24 Halbwörter
 D-Bereich = --
 Indezzellen = 6 (Adressen 0...5)

Bild 2.8b Angaben zum Unterprogramm aus Bild 2.8a

Der Versorgungsblock wurde in Form von Konstanten angegeben. Sie wurden somit im K-Bereich abgelegt. Sollen sie veränderlich sein, so ist jeder Konstanten noch ein V anzufügen. Alle Versorgungsparameter haben Halbwortlänge. Im Bild 2.9 ist gezeigt, wie der Versorgungsblock aus Bild 2.8a auch als Literal des Befehls BA geschrieben werden kann.

Soll der Versorgungsblock im B- oder D-Bereich liegen, so kann das Register A nicht mit dem Befehl BA vorbelegt werden. Es kann dann der Befehl B2 verwendet werden. Im Bild 2.10 ist das Beispiel aus Bild 2.8a abgewandelt, indem der Versorgungsblock in den D-Bereich gelegt wurde.

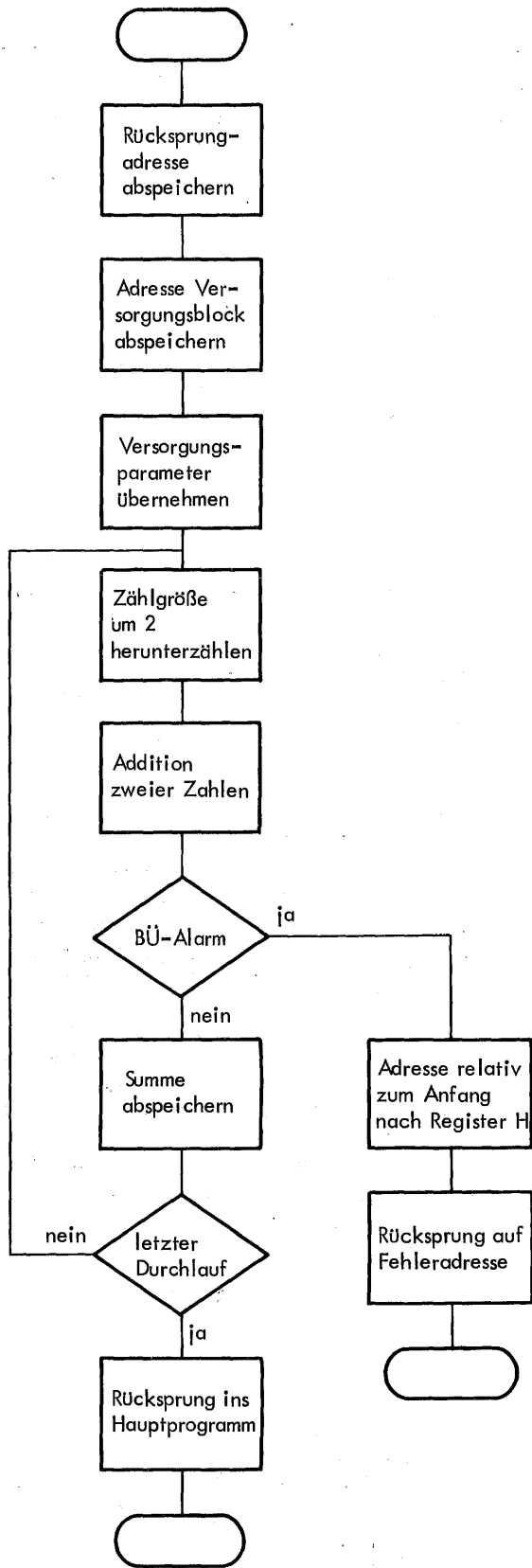


Bild 2.8c Flußdiagramm zum Bild 2.8a

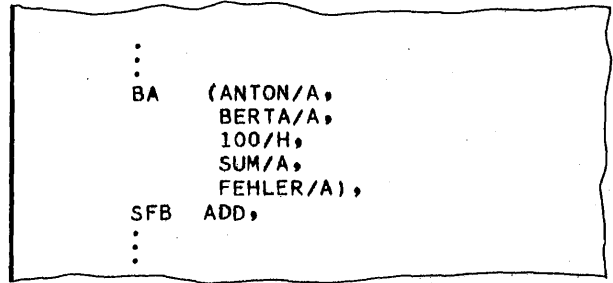


Bild 2.9 Versorgungsblock im Literal

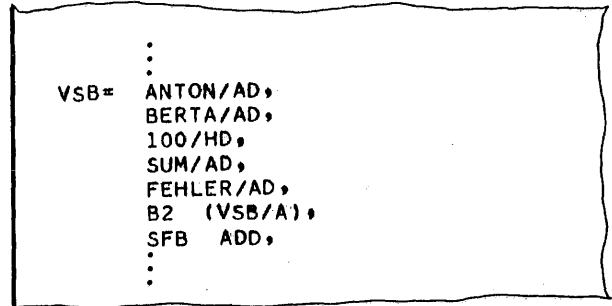


Bild 2.10 Versorgungsblock im D-Bereich

Im Unterprogramm Bild 2.8a wurde zuerst die technische Rücksprungadresse aus dem Register B in der Indexzelle TRX sichergestellt und danach die Adresse des Versorgungsblocks aus dem Register A in der Indexzelle XVSB. Diese Sicherstellung kann natürlich auch in einer Speicherzelle vorgenommen werden. Es müssen dann die entsprechenden Befehle verwendet werden.

Danach wurden die Versorgungsparameter in Indexzellen übernommen. Dazu dient die Befehlsfolge

$$\left. \begin{array}{l} \text{XB} \quad \text{XVSB,} \\ \text{MAB} \quad \text{TCB } p, \\ \text{XC} \quad i, \end{array} \right\} \text{ergibt: } \left\{ \begin{array}{l} \text{TCB} \quad \text{VSB}+p, \\ \text{XC} \quad i, \end{array} \right.$$

Dies ist dann von Vorteil, wenn die Versorgungsparameter innerhalb einer Schleife oft benötigt werden. Es ist jedoch auch möglich, die Versorgungsparameter nicht in Indexspeicherzellen zu übernehmen, sondern direkt aus dem Versorgungsblock. Im Beispiel Bild 2.8a wurde dies für den letzten Versorgungsparameter, der die Fehleradresse enthält, mit der Befehlsfolge

$$\left. \begin{array}{l} \text{XB} \quad \text{XVSB,} \\ \text{MAB} \quad \text{SE } 4, \end{array} \right\} \text{ergibt} \quad \text{SE } \langle \text{XVSB} \rangle + 4,$$

entspricht SE VSB+4,

durchgeführt. Da die Fehleradresse nur in seltenen Fällen benötigt wird, ist dieser Weg in diesem Fall zweckmäßig.

Hauptprogramm

```

:
:
VSB= ANTON/A,
      BERTA/A,
      100/H,
      SUM/A,
      FEHLER/A,
      SFB ADD,
TR=  NULL VSB,
:
:

```

Unterprogramm

```

ADD.= SEGM,
XC   TRX,
E    TCB TRX,
XC   XVERS,
:
:
XB   TRX,
MAB S 1,

```

technische Rücksprungadresse
in Indexzelle sicherstellen

Unterprogramm

```

ADD.= SEGM,
TBC  TRS,
MAB  TCB 0,
XC   XVSB,
:
:
MA   1,
SE   TRS,
TRS= ASP 1,

```

technische Rücksprungadresse
in Speicherzelle sicherstellen

Bild 2.11 Beispiel Adresse des Versorgungsblocks hinter Befehl SFB

2.3. Adresse des Versorgungsblocks hinter SFB

Die Anfangsadresse des Versorgungsblocks kann auch hinter (oder vor) dem Befehl stehen, mit dem in das Unterprogramm gesprungen wurde. In Bild 2.11 ist das Beispiel aus Bild 2.8a entsprechend abgewandelt. Der Versorgungsblock kann bei der Niederschrift zwischen den Befehlen stehen, da das Programm beim Assemblieren in den K-, V-, B- und D-Bereich getrennt wird. Im Beispiel kommt der Versorgungsblock in den K-Bereich. Die Anfangsadresse dieses Versorgungsblocks ist eine 16-Bit-Adresse und kann im Adressenteil des Befehls NULL stehen. Damit wird die Anfangsadresse im B-Bereich (bei den Befehlen) abgelegt. Es ist jedoch auch möglich und bei 24-Bit-Adressen erforderlich,

$$TR = VSB/AB,$$

zu schreiben. Es muß die Spezifikation B hinzugefügt werden. Solange die Anfangsadresse des Versorgungsblocks nicht länger als 16 Bits ist (Versorgungsblock im K- oder V-Bereich), kann auf die technische Rücksprungadresse zurückgesprungen werden. Unter dieser Adresse steht der Befehl NULL, der keine Wirkung hat. In den

beiden Unterprogrammen (Bild 2.11) könnte dann mit den Befehlen

E S TRX, bzw. SE TRS,

zurückgesprungen werden.

2.4. Versorgungsblock hinter SFB

Der Versorgungsblock kann auch direkt hinter dem Befehl SFB stehen. Damit sind die Anfangsadresse des Versorgungsblocks und die technische Rücksprungadresse identisch. Es braucht also nur eine Adresse abgespeichert zu werden. Der Versorgungsblock muß jedoch so aufgebaut sein, daß er im B-Bereich verbleibt, also ggf. müssen die Versorgungsparameter mit der Spezifikation B versehen werden. Sie sind damit alle schreibgeschützt und können per Programm nicht verändert werden. Wenn einer der Versorgungsparameter eine 24-Bit-Größe ist, so muß auf jeden Fall hinter den Versorgungsblock zurückgesprungen werden. Ansonsten gilt das unter Abschnitt 2.2. und 2.3. Gesagte.

Hauptprogramm

```

:
:
TR=  SFB ADD,
      ANTON/AB,
      BERTA/AB,
      100/HB2,
      SUM/AB,
      FEHLER/AB,
:
:

```

Unterprogramm

```

ADD.= SEGM,
XC   TRX,
E    TCB TRX,
XC   XANTON,
MA   1,
E    TCB TRX,
XC   XBERTA,
:
:
MA   5,
E    S TRX,
:
:
MA   4,
E    S TRX,

```

techn. Rücksprungadresse
sicherstellen

Übernahme der Parameter

Rückkehr

Rückkehr nach Fehler

Bild 2.12 Versorgungsblock hinter SFB

Diese Art der Parameterübergabe ist besonders geeignet für den Aufruf der Unterprogramme mit Hilfe des Befehls SU oder SUE, wie es im Abschnitt 3. beschrieben ist.

Im Bild 2.12 ist gezeigt, wie das Beispiel aus Bild 2.8a abzuändern ist. Wenn die Parameter 16-Bit-Größen sind, können sie auch – wie im Bild 2.13 gezeigt – im Adressenteil des Befehls NULL (kann auch mit N gekürzt geschrieben werden) untergebracht werden. In diesem Fall kann die Angabe einer Spezifikation entfallen; da es Befehle sind, werden sie im B-Bereich abgelegt.

```

:
:
:
TR= SFB ADD,
      NULL ANTON,
      NULL BERTA,
      NULL 100,
      NULL SUM,
      NULL FEHLER,
:
:

```

Bild 2.13 Parameter im Nullbefehl

2.5. Versorgungsbefehle

Der Versorgungsbereich kann auch Versorgungsbefehle enthalten. Diese Befehle kommen mit Hilfe des Befehls T zur Ausführung.

In Bild 2.14 ist das Beispiel aus Bild 2.8a abgewandelt. Der Versorgungsbereich enthält Versorgungsbefehle, die im Unterprogramm durch den Befehl T zur Ausführung kommen. Alle Versorgungsbefehle bringen den Versorgungsparameter in das Register B.

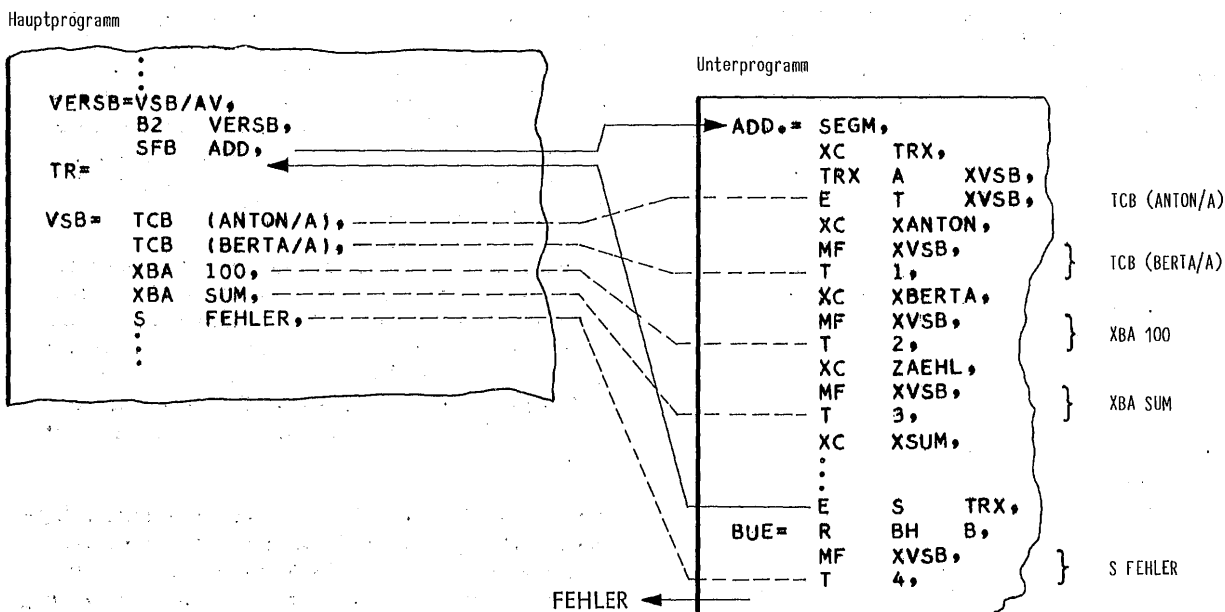


Bild 2.14 Beispiel für Versorgungsbefehle im Versorgungsbereich (Abwandlung des Beispiels aus Bild 2.8a)

Im Beispiel können die ersten beiden Parameter 24-Bit-Größen sein, während bei den anderen nur 16-Bit-Größen möglich sind. Da im Hauptprogramm bekannt ist, wie groß die Parameterwerte sind, kann der entsprechende Versorgungsbefehl ausgewählt werden. Ist nun z.B. das Programmstück "FEHLER" nicht in der Großseite 1, so kann der Versorgungsbefehl

```
SE (FEHLER/A),
```

lauten. Damit kann in eine andere Großseite gesprungen werden.

Das Unterprogramm muß so ausgelegt sein, daß es 24-Bit-Größen verarbeiten kann. Werden die Versorgungsparameter in Index- oder Speicherzellen übernommen, so ist dies gewährleistet. Neben den in Bild 2.14 angegebenen Beispielen gibt es weitere Möglichkeiten. So kann z.B. mit dem Unterprogramm vereinbart sein, daß die Zählgröße in der Indexzelle 5 liegt. Der Versorgungsbefehl XBA 100 könnte dann ersetzt werden von

```
ZX 100 5,
```

wodurch im Unterprogramm der Befehl XC ZAEHL entfallen könnte.

2.6. Versorgungsbereich in Indexzellen

Die Versorgungsparameter können den Unterprogrammen auch über Indexzellen zugänglich gemacht werden.

Einmal können es die Indexzellen sein, die für die gleichen Parameter von Haupt- und Unterprogrammen gemeinsam benutzt werden.

Zum anderen können die vom Unterprogramm für die Parameter festgelegten Indexzellen vom Hauptprogramm mit dem Parameter belegt werden. Dies kann z.B. mit dem Befehl TXX $i_L i_R$ oder mit der Befehlsfolge TCB m , XC i erfolgen. Anschließend wird zum Unterprogramm gesprungen.

D

TAS - Handbuch

Dez. 68

Hauptprogramm

```

:
:
TXX 0 S1,
TXX 1 S2,
ZX 100 2,
XBA (S= ASP 200/G),
XC 3,
TCB (FEH/A),
XC 4,
SFB ADD,
TR=
:

```

Unterprogramm

```

ADD.= SEGM,
INDEX 0(ANTON,BERTA,ZAEHL,
SUM,FEHLER,TRX),
XC TRX,
ANF= HXP -2 ZAEHL,
M ZAEHL,
E B ANTON,
EMB A BERTA,
SAA BUE,
EMB C SUM,
SXN ANF,
E S TRX,
BUE= R BH B,
E S FEHLER,

```

Parameterversorgung für Unterprogramm ADD	
Indezzelle	0 : Anfangsadresse Summand 1
	1 : Anfangsadresse Summand 2
	2 : Anzahl der Durchläufe mal 2 (Anzahl der Ganzwörter mal 2)
	3 : Anfangsadresse für die Summe
	4 : Fehleradresse bei Bereichsüberschreitung

Bild 2.15 Übergabe der Versorgungsparameter in Indexzellen (Abwandlung aus Beispiel Bild 2.8a)

Im Bild 2.15 ist das Beispiel aus Bild 2.8a abgewandelt. Die Übergabe der Parameter wird vom Unterprogramm in den Indexzellen 0 bis 4 verlangt.

Die Indexbasis kann sowohl vom Hauptprogramm als auch vom Unterprogramm neu gesetzt bzw. umgeschaltet werden.

Das Beispiel im Bild 2.16 zeigt eine Indexbasisumschaltung für das Unterprogramm, wobei die Umschaltung im Hauptprogramm durchgeführt wird. Im Unterprogramm werden 6 Indexzellen benötigt, im Hauptprogramm werden daher mit ASP 6 Halbwörter dafür freigehalten. In diesem Fall braucht das Unterprogramm den Bedarf an Indexzellen nicht zu berücksichtigen.

2.7. Umschalten Indexbasis

Jedes Unterprogramm kann einen eigenen Indexbereich bekommen. Dies ist besonders bei sehr umfangreichen Unterprogrammen von Bedeutung. Zu diesem Zweck wird das Indexbasisregister (Register X) mit den Befehlen BCI oder ZI auf einen neuen Wert gesetzt. Einzelheiten zum Umschalten der Indexbasis sind in dem Abschnitt 5.3. Befehle und Adressierung gegeben. An dieser Stelle soll auf die Anwendung bei Unterprogrammen eingegangen werden.

Im Beispiel Bild 2.17 wurde die Umschaltung im Hauptprogramm mit dem Befehl ZI vorgenommen. Nach dem Rücksprung muß die Indexbasis wieder zurückgeschaltet werden. Damit die Indexbasis zum Zurückschalten auf den alten Wert bekannt ist, wurde sie zu Anfang des Programms auf einen definierten Wert (XBER) gesetzt. Es ist jedoch auch möglich, die aktuelle Indexbasis mit dem Befehl BLEI 4 festzustellen (siehe dazu Abschnitt 5.3.2. Befehle und Adressierung). Die Umschaltung wäre auch mit dem Befehl BCI möglich. Dies ist am Beispiel im Bild 2.18 gezeigt.

Hauptprogramm

```

:
:
X= ASP 6,
BCI (A=X/AV,0/HV),
SFB UPR,
TR= BCI A,
:
:

```

Bild 2.16 Umschalten der Indexbasis für das Unterprogramm im Hauptprogramm

In den vorstehenden beiden Fällen brauchte keine Reservierung von Halbwörtern für die Indexzellen vorgenommen zu werden, da die Indexzellen bereits in dem alten Indexbereich vorhanden waren und durch die Umschaltung ein neuer Indexbereich innerhalb des alten Indexbereichs aufgemacht wurde. Der neue Indexbereich hat 256 Indexzellen, jedoch werden nur die ersten 6 verwendet.

Das Beispiel im Bild 2.19 zeigt eine Indexbasisumschaltung im Unterprogramm. Es ist eine Variante des Beispiels aus Bild 2.8a. Für die 6 benötigten Indexzellen werden durch ASP 6 Halbwörter freigehalten. Im Anfang des Unterprogramms wird die Indexbasis auf diese 6 Zellen umgeschaltet. Noch bevor die erste Indexzelle an-

gesprochen wird, muß die Indexbasis umgeschaltet sein. Vor dem Rücksprung in das Hauptprogramm wird die Indexbasis wieder zurückgeschaltet. Beim Rücksprung muß beachtet werden, daß nach dem Umschalten der Indexbasis die Indexzellen nicht mehr zur Verfügung stehen; im Beispiel also auch nicht mehr die Indexzelle, in der die technische Rücksprungadresse steht. Sie wird daher vorher in das Register B gebracht. Der Bedarf an Indexzellen im Unterprogramm braucht im Hauptprogramm nicht berücksichtigt zu werden.

Hauptprogramm

```

      SEGM,
      ZI (XBER/AV),
XBER.=ASP 50/G,
      :
      :
      INDEX 35(S1,S2,Z,S,F,1),
      :
      :
      ZX 100 Z,
      XBA FEH,
      XC F,
      ZI (XBER + 35/AV),
      SFB ADD,
TR=  ZI (XBER/AV),
      :
      :

```

Indexzuordnungen

Hauptprogramm (Indexbasis XBER)		Unterprogramm (Indexbasis XBER +35)	
symbolisch	absolut	symbolisch	absolut
S1	35	ANTON	0
S2	36	BERTA	1
Z	37	ZAEHL	2
S	38	SUM	3
F	39	FEHLER	4
-	40	TRX	5

Bild 2.17 Umschalten der Indexbasis für das Unterprogramm im Hauptprogramm (Unterprogramm gemäß Bild 2.15)

Hauptprogramm

```

      SEGM,
      :
      :
      INDEX 35(S1,S2,Z,S,F,1),
      :
      :
      ZX 100 Z,
      XBA FEH,
      XC F,
      BLEI 4,
      SH AU 24,
      AA 35,
      SH ALU 24,
      C A,
A=  ASP 2/G,
      BCI A,
      SFB ADD,
TR=  BCI A,
      :
      :

```

Bild 2.18 Variation des Beispiels aus Bild 2.17

Unterprogramm

```

ADD.= SEGM,
      INDEX 0(TRX,XVSB,XANTON,
              XBERTA,ZAEHL,XSUM),
X=  ASP 6,
      BCI (A=X/AV,0/HV),
      XC TRX,
      :
      :
      XB TRX,
      BCI A,
      MAB S 0,
BUE= R BH B,
      XB XVSB,
      BCI A,
      MAB SE 4,

```

Bild 2.19 Umschalten der Indexbasis im Unterprogramm (Hauptprogramm gemäß Bild 2.8a)

D

TAS - Handbuch

Dez. 68

3. UNTERPROGRAMMAUFRUF ÜBER SU ODER SUE

Mit Hilfe des Unterprogrammregisters (Register U) ist eine spezielle Organisation der Unterprogramme möglich. Für den Aufruf des Unterprogramms stehen dazu die Befehle SU und SUE zur Verfügung. Des weiteren sind, speziell auf diese Befehle abgestimmt, die Doppelcodebefehle MU und EMU vorhanden.

Beim Aufruf eines Unterprogramms mit den Befehlen SFB bzw. SFBE (siehe Abschnitt 2.) wird die technische Rücksprungadresse (d.h. die Adresse des Befehls, der auf den Befehl SFB bzw. SFBE folgt) dem Unterprogramm im Register B zur Verfügung gestellt, und dieses muß die technische Rücksprungadresse sicherstellen. Beim Befehl SU bzw. SUE speichert dagegen das Hauptprogramm die technische Rücksprungadresse ab und stellt sie damit sicher. Dies ist in der Wirkung der Befehle SU und SUE mit enthalten. Das Unterprogramm braucht also eine Sicherstellung nicht vorzunehmen. Das über den Aufruf der Unterprogramme mit Hilfe der Befehle SFB und SFBE Beschriebene, gilt weitgehend auch für die Befehle SU und SUE. Abweichungen und Änderungen gegenüber SFB und SFBE sind jeweils aufgeführt.

Das Hauptprogramm speichert die technische Rücksprungadresse in einer Indexzelle ab. In welcher Indexzelle sie abgespeichert werden soll, muß vorher angegeben werden. Dazu steht der Pseudobefehl

UNTPR i, i: Indexadresse

zur Verfügung. Er bewirkt, daß vor dem Start des Operators das Unterprogrammregister für die Abspeicherung der technischen Rücksprungadresse auf die durch "i" angegebene Indexadresse gesetzt wird. Der Pseudobefehl UNTPR wird aber für einen Operator nur einmal zur Festlegung der Indexzelle angewendet, und zwar muß er zu Beginn eines der vorhandenen Quellenprogramme stehen. Soll kein Sprung in ein Unterprogramm mit SU bzw. SUE erfolgen, so ist das Register U zweckmäßigerweise am Anfang auf 254 zu setzen. Tritt danach der Befehl SU bzw. SUE auf, so wird dieser Befehl zwar ausgeführt, aber es wird gleichzeitig eine Alarmmeldung (U-Alarm) an das Betriebssystem gegeben. Der weitere Ablauf hängt von der Behandlung dieses Alarms ab.

Soll im weiteren Verlauf des gleichen Programms oder der anderen Quellenprogramme eine andere Indexzelle für die Abspeicherung der technischen Rücksprungadresse festgelegt werden, so muß mit dem Befehl ZU das Unterprogrammregister entsprechend verändert werden.

ZU	i	Setze Unterprogrammregister	<U> := i
----	---	-----------------------------	----------

Quellenprogramm 1

```

SEGMENT,
:
UNTPR X1-1,
INDEX X1(1),
:
:
:
:
ENDE,
    
```

Indexzelle X1-1 im Register U

Quellenprogramm 2

```

SEGMENT,
ZU 240,
:
:
:
:
ENDE,
    
```

Indexzelle 240 im Register U

Bild 3.1 Beispiele für das Setzen des Registers U

Der Befehl ZU enthält im Adressenteil die Adresse einer Indexzelle. Diese Adresse wird in das Register U (Unterprogrammregister) gebracht. Bei einem Unterprogramm-sprung mit dem Befehl SU oder SUE wird diese (durch ZU bzw. UNTPR festgelegte) Indexadresse im Register U um 1 erhöht. In diese (um 1 erhöhte) Indexzelle wird die technische Rücksprungadresse abgespeichert. Soll also die technische Rücksprungadresse in der Indexzelle TRX abgespeichert werden, so muß das Register U auf den Wert TRX-1 gesetzt werden.

Soll die technische Rücksprungadresse in der Indexzelle 0 abgespeichert werden, so muß das Register U auf 255 gesetzt werden. Dies ist die maximal erlaubte Adresse. Bei einer Erhöhung um 1 entsteht die Indexadresse 0.

```

ZU 14,
ZU 255,
ZU X5,
ZU TRX-1,
    
```

Bild 3.2 Beispiele für den Befehl ZU

D

Wird in einem Programm der Befehl SU oder SUE verwendet, so muß in jedem Fall vorher das Register U gesetzt werden. Es kann sonst einen beliebigen Stand haben, und der Inhalt einer wichtigen Indexzelle könnte zerstört werden.

Die Abspeicherung der technischen Rücksprungadresse erfolgt nicht in die mit ZU bzw. UNTPR angegebene Indexzelle, sondern in die nächst höhere. Es ist deshalb darauf zu achten, daß diese Indexzelle frei bleibt und vom Haupt- oder Unterprogramm nicht anderweitig belegt wird. In Bild 3.3 ist dies an mehreren Beispielen gezeigt.

Wird das Register U auf den Wert 255 gesetzt, so ist als nächste die Indexzelle 0 zu reservieren. Folgt nun der Befehl SU, so wird die technische Rücksprungadresse in dieser freigehaltenen Indexzelle abgelegt.

Werden im Programm nicht alle 256 Indexzellen belegt, so kann das Register U auf eine Indexzelle gesetzt werden, die bestimmt nicht mehr vom übrigen Programm benutzt wird (z.B. die Indexzelle 250). Damit stehen die folgenden Indexzellen zur Verfügung. Sind mehrere Unterprogrammstufen (siehe Abschnitt 1.2.) vorhanden, so sind entsprechend viele Indexzellen freizuhalten, um die jeweiligen technischen Rücksprungadressen aufzunehmen.

Im letzten Beispiel im Bild 3.3 sind die Indexzellen XA, XB, XC für das übrige Programm reserviert. Da die technische Rücksprungadresse nicht in der im Register U stehenden Indexzelle abgelegt wird, sondern in der darauffolgenden, kann in diesem Beispiel das Register U auf XC gesetzt werden. Die darauffolgende Indexzelle wird für die technische Rücksprungadresse reserviert. Die Indexzelle XC ist global und behält ihren Geltungsbereich auch für andere Segmente.

Hauptprogramm

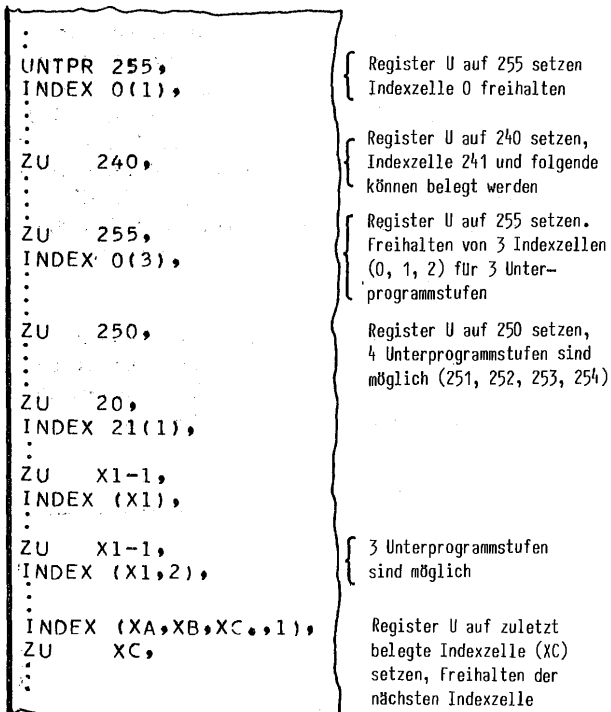


Bild 3.3 Beispiele für Setzen des Registers U und Freihalten der nächsten Indexzellen

Der Sprung ins Unterprogramm erfolgt mit dem Befehl SU bzw. SUE.

SU	m	Springe in Unterprogramm	$\langle U \rangle := \langle U \rangle + 1$ $\langle \langle U \rangle \rangle := \langle F \rangle + 1$ $\langle F \rangle_{9-24} := m$
----	---	--------------------------	---

Bei dem Sprungbefehl SU steht im Adressenteil die Adresse bzw. der Name des Unterprogramms, in das gesprungen werden soll. Der Befehl erhöht zuerst die Indexadresse, die im Register U steht, um 1. Danach wird in die Indexzelle, deren Adresse nun im Register U steht, die technische Rücksprungadresse abgespeichert, d.h. die Adresse des Befehls SU um 1 erhöht oder mit anderen Worten, die Adresse des auf SU folgenden Befehls. Danach wird der Befehlsfolgezähler (Register F) auf die im Befehl SU angegebene Adresse gesetzt. Dies ist die Adresse des Unterprogramms, und es wird im Unterprogramm fortgefahren.

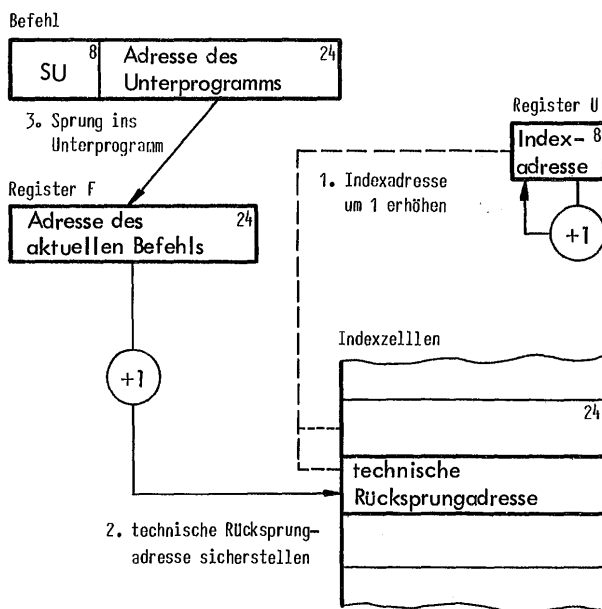


Bild 3.4 Unterprogrammierung mit Befehl SU

SUE	m	Springe in Unterprogramm nach Ersetzung	$\langle U \rangle := \langle U \rangle + 1$ $\langle \langle U \rangle \rangle := \langle F \rangle + 1$ $\langle F \rangle := \langle m \rangle + \text{mod} 2$
-----	---	---	---

Der Befehl SUE hat die gleiche Wirkung wie der Befehl SU, nur steht im Adressenteil des Befehls SUE nicht die Adresse des Unterprogramms, sondern die Adresse eines Halbwortes, in dem die Adresse des Unterprogramms

steht. In diesem Fall kann also die Adresse eine 24-Bit-Größe sein. Mit dem Befehl SUE kann auch in eine andere Großseite gesprungen werden. (Beim Befehl SU ist der Sprung in eine andere Großseite nur möglich, wenn er mit dem Befehl MABI modifiziert wurde). Das in folgenden Abschnitten über den Sprung ins Unterprogramm mit dem Befehl SU geschriebene gilt entsprechend auch für den Befehl SUE.

```

SU  UP,
ADRUP=UP/A,
SUE ADRUP,
SUE (UP/A),

```

Bild 3.5 Beispiel für die Befehle SU und SUE

Hat p einen von 0 verschiedenen Wert, so kann um p Befehle vor oder hinter der technischen RückspRUNG-adresse gesprungen werden. Dies ist von besonderer Bedeutung, wenn hinter dem Befehl SU ein Versorgungsblock liegt (siehe die folgenden Abschnitte). Des weiteren ist es möglich, daß das Unterprogramm zwei Ausgänge hat und entsprechend dem Ergebnis auf eine bestimmte Stelle im Hauptprogramm springt.

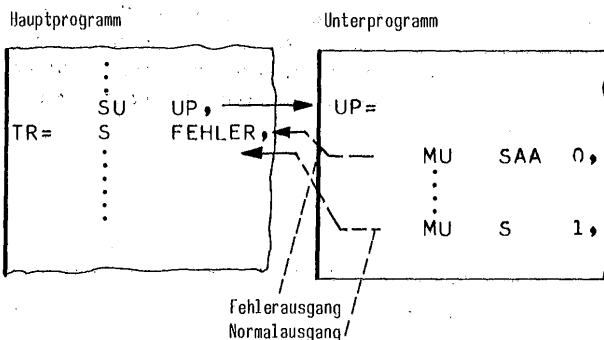


Bild 3.7 Beispiel für 2 Ausgänge

Der Rücksprung ins Hauptprogramm erfolgt stets mit dem Doppelcodebefehl (Ersetzbefehl) MU, der als Zweitcode jeden Sprungbefehl enthalten kann, der im Adressenteil ein Halbwort adressiert (Buchstabe m im Adressenteil). Ein Sprungbefehl mit relativer Sprungadresse (Buchstabe p im Adressenteil) kann nicht verwendet werden.

MU	c	p	Modifiziere über U	adr := <<U>> + p
----	---	---	--------------------	------------------

Der Befehl MU bewirkt, daß ein Zweitbefehl erzeugt wird, dessen Adressenteil aus der technischen RückspRUNG-adresse besteht. Zu dieser technischen RückspRUNG-adresse wird noch der Wert p addiert. Ist der Zweitbefehl ein unbedingter Sprungbefehl oder ein bedingter Sprungbefehl, dessen Sprungbedingung erfüllt ist, so wird bei p = 0 auf die technische RückspRUNG-adresse zurückgesprungen. Ein Sprungbefehl der bei MU als Zweitcode auftritt, kann auch in eine andere Großseite springen.

```

MU  S  0,
MU  SXG 5,
MU  SN 1,

```

Bild 3.6 Beispiel für Rückkehr aus dem Unterprogramm

Wird der Sprung als Zweitbefehl beim Befehl MU ausgeführt, so wird gleichzeitig der Inhalt des Registers U um 1 vermindert. Damit ist der Stand vor dem Befehl SU wieder hergestellt. Für einen späteren Sprung in ein Unterprogramm wird also wieder die gleiche Indexzelle benutzt wie beim vorhergehenden.

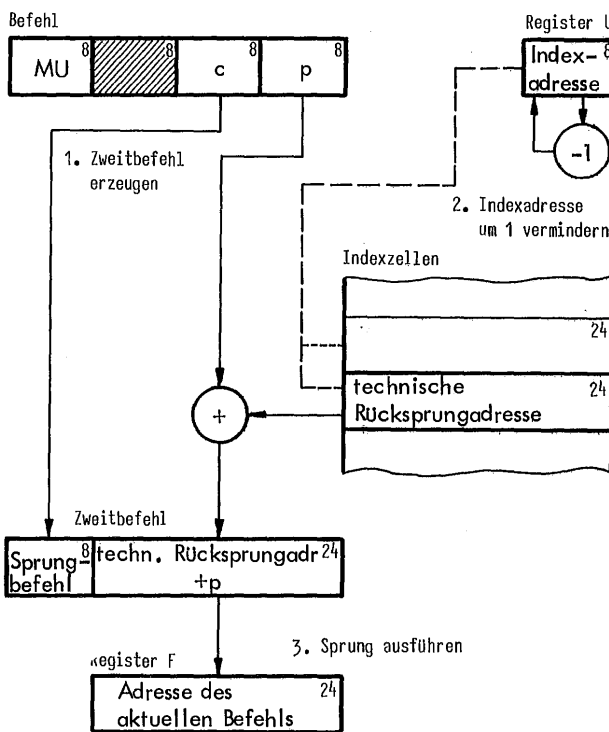


Bild 3.8 Rücksprung mit dem Befehl MU

TAS - Handbuch

Dez. 68

D

Wird von einem Unterprogramm in ein weiteres gesprungen, werden also mehrere Unterprogrammstufen, wie in Abschnitt 1.2. beschrieben, verwendet, so gilt bei einem Unterprogrammssprung mit dem Befehl SU oder SUE das vorstehend Gesagte. Es wird also bei jedem Befehl SU oder SUE das Register U um 1 hochgezählt; das bedeutet, daß die jeweilige technische Rücksprungadresse in aufeinanderfolgenden Indexzellen abgespeichert wird. Bei x Unterprogrammstufen müssen also x Indexzellen zur Aufnahme der technischen Rücksprungadresse im Hauptprogramm (I) freigehalten werden. In den Unterprogrammen brauchen keine Rücksprungadressen sichergestellt zu werden. In Bild 3.9 ist an Beispielen das Reservieren dieser Indexzellen für 5 Unterprogrammstufen gezeigt. (An Stelle des Befehls ZU könnte in den Beispielen auch der Befehl UNTPR stehen.)

```

INDEX 16(5),
ZU 15,

INDEX 6(5),
ZU 255,

INDEX (X1,X2,X3,5),
ZU X3,

```

Bild 3.9 Beispiele zum Reservieren von Indexzellen für 5 Unterprogrammstufen

Der Rücksprung erfolgt in der umgekehrten Reihenfolge des Aufrufs, wobei jedesmal die Indexadresse im Register U, wie beschrieben, um 1 vermindert wird.

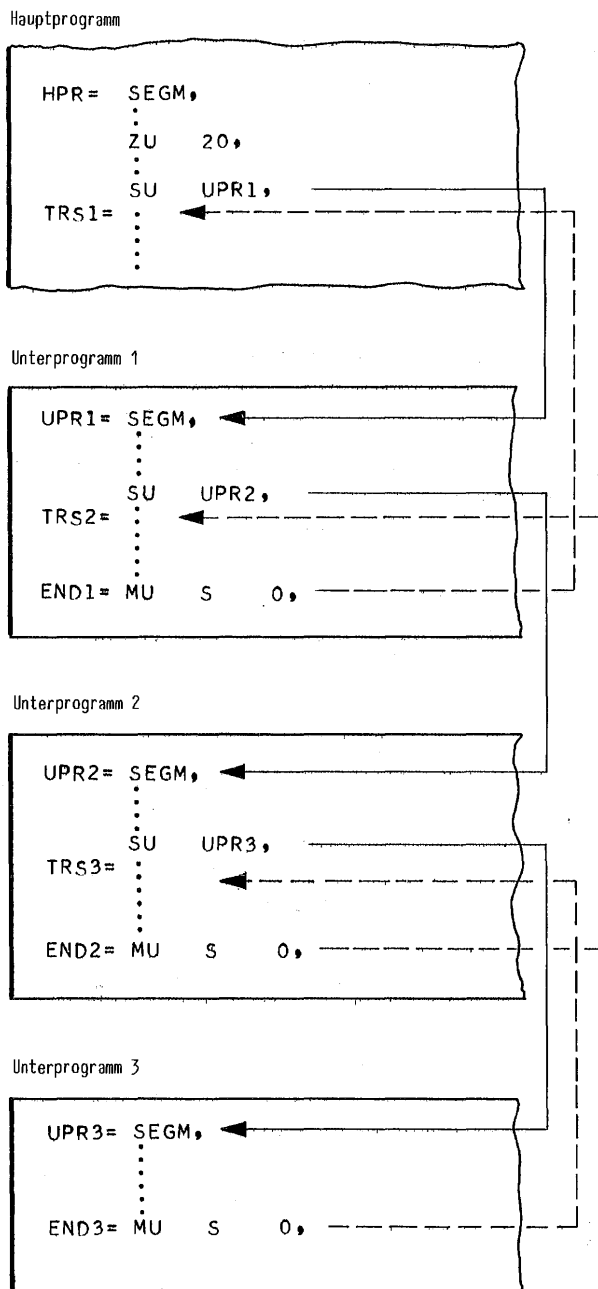


Bild 3.10 Prinzip der mehrstufigen Unterprogramme

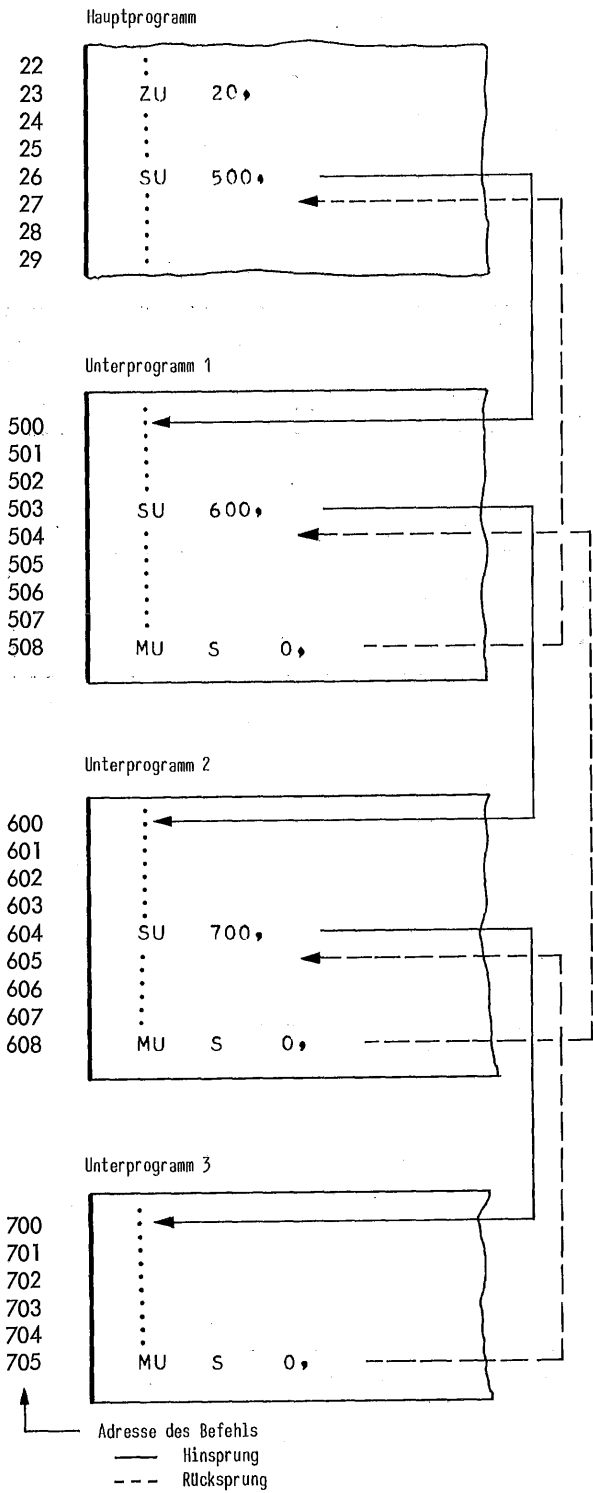


Bild 3.11 Beispiel aus Bild 3.10 nach dem Assemblieren

Reg. F	Befehl	Programm	Reg. U	Indezelle			
				20	21	22	23

22	:	Hauptprogramm	↓				
23	ZU 20,		20				
24	:		↓				
25	:		↓				
26	SU 500,		21		27		
500	:	Unterprogramm 1	↓				
501	:		↓				
502	:		↓				
503	SU 600,		22			504	
600	:	Unterprogramm 2	↓				
601	:		↓				
602	:		↓				
603	:		↓				
604	SU 700,		23				605
700	:	Unterprogramm 3	↓				
701	:		↓				
702	:		↓				
703	:		↓				
704	:		↓				
705	MU S 0,		22				
605	:	Unterprogramm 2	↓				
606	:		↓				
607	:		↓				
608	MU S 0,		21				
504	:	Unterprogramm 1	↓				
505	:		↓				
506	:		↓				
507	:		↓				
508	MU S 0,		20				
27	:	Hauptprogramm	↓				
28	:		↓				
29	:		↓				

Bild 3.12 Ablauf des Beispiels in Bild 3.11

Die Versorgung des Unterprogramms mit den nötigen Daten geschieht über die Versorgungsparameter. Sie umfassen sowohl Angaben über die Eingangsdaten als auch über die Ausgangsdaten (vom Unterprogramm aus gesehen). Die Übergabe der Versorgungsparameter kann auf verschiedene Arten erfolgen. In den nachfolgenden Abschnitten sind die wichtigsten Möglichkeiten aufgezeigt. Die verschiedenen Möglichkeiten können auch - soweit sinnvoll - miteinander kombiniert werden.

D

3.1. Versorgungsparameter im Register

Einzelne wenige Daten können, wie in Abschnitt 2.1. bereits beschrieben, dem Unterprogramm in den Registern übergeben werden.

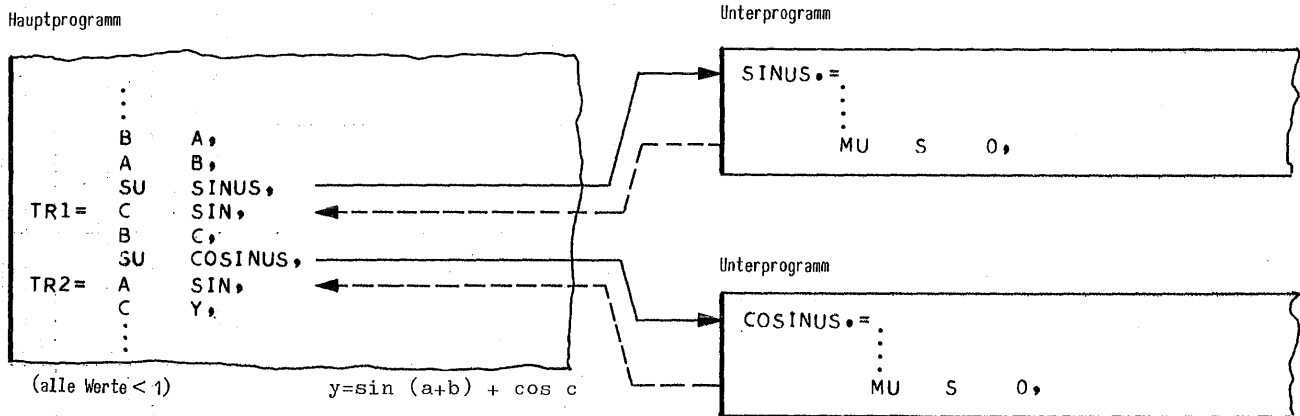


Bild 3.13 Beispiel für Versorgungsparameter im Register

Im Bild 3.13 wurde das Beispiel aus Bild 2.5 auf den Befehl SU abgewandelt. Es entfällt im Unterprogramm das Sicherstellen der technischen Rücksprungadresse.

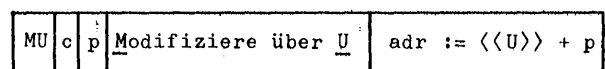
3.2. Versorgungsblock hinter SU

Es ist besonders günstig, den Versorgungsblock direkt hinter (oder vor) den Befehl SU zu legen, da die Übernahme der Parameter ins Unterprogramm mit den dafür vorgesehenen Befehlen MU oder EMU erfolgen kann.

Es muß dafür gesorgt werden, daß der Versorgungsblock im Befehlsbereich liegt. Die Parameter sind mit der Spezifikation B zu versehen und können dann bis zu 24 Bits lang sein. Sie sind damit schreibgeschützt und können per Programm nicht verändert werden.

Sind die Parameter 16-Bit-Größen, so können sie auch im Adressenteil des Befehls NULL untergebracht werden. (Dies ist aus Bild 3.19 ersichtlich.)

Für das Holen der Versorgungsparameter, die die Halbwörter vor oder hinter dem Unterprogrammssprung (SU) belegen, sind die Ersetzbefehle MU und EMU vorgesehen (siehe auch Abschnitt "Adressenrechnung", 10.3.). Beide Befehle erzeugen einen Zweitbefehl.



$p: \pm 0 \dots \pm 127$
 $\langle U \rangle := \langle U \rangle - 1$
 wenn c Sprungbefehl
 und Sprungbedingung
 erfüllt ist

Der Zweitbefehl erhält beim Befehl MU im Adressenteil die auf SU folgende Adresse (technische Rücksprungadresse), erhöht um den Wert p. Der Wert p kann die Werte ± 0 bis ± 127 annehmen. Das bedeutet, daß vor dem Befehl SU 126 Halbwörter (Versorgungsparameter) und hinter dem Befehl SU 128 Halbwörter (Versorgungsparameter) erreicht werden können.

EMU	c	p	Ersetze nach Modifizierung über <u>U</u>	adr := <<U>>+p
-----	---	---	---	----------------

p: ± 0... ±127

Während der Befehl MU die Adresse eines Versorgungsparameters in der Nähe des Befehls SU dem Zweitbefehl zur Verfügung stellt, wird beim Befehl EMU nochmals eine Ersetzung vorgenommen. Dies bedeutet, daß der Inhalt einer Zelle vor oder hinter dem Befehl SU, also der Versorgungsparameter selbst, als Adressenteil des Zweitbefehls verwendet wird.

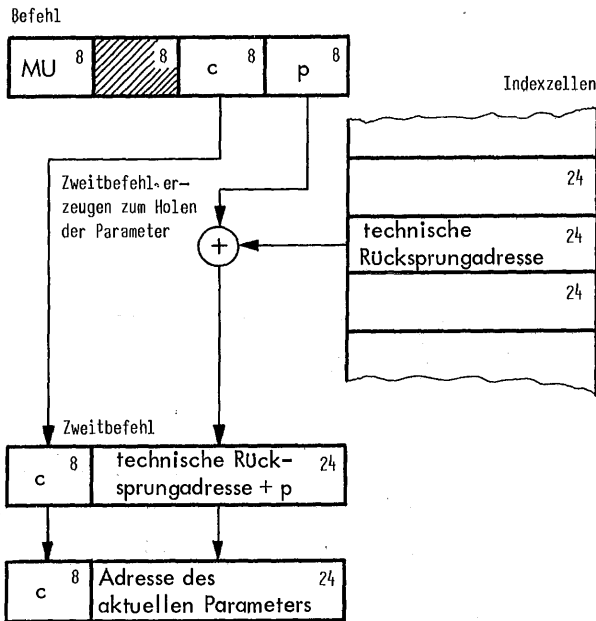


Bild 3.14 Holen der Versorgungsparameter über den Befehl MU

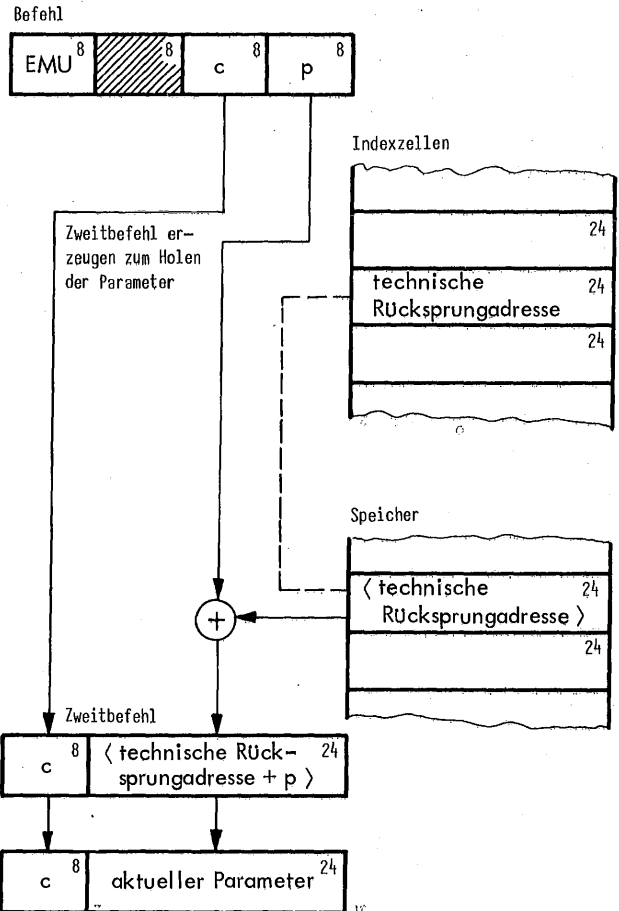


Bild 3.15 Holen der Versorgungsparameter über den Befehl EMU

D

TAS - Handbuch

Dez. 68

Der Rücksprung vom Unterprogramm ins Hauptprogramm erfolgt immer mit dem Befehl MU hinter die Versorgungsparameter, wenn diese hinter dem Befehl SU stehen.

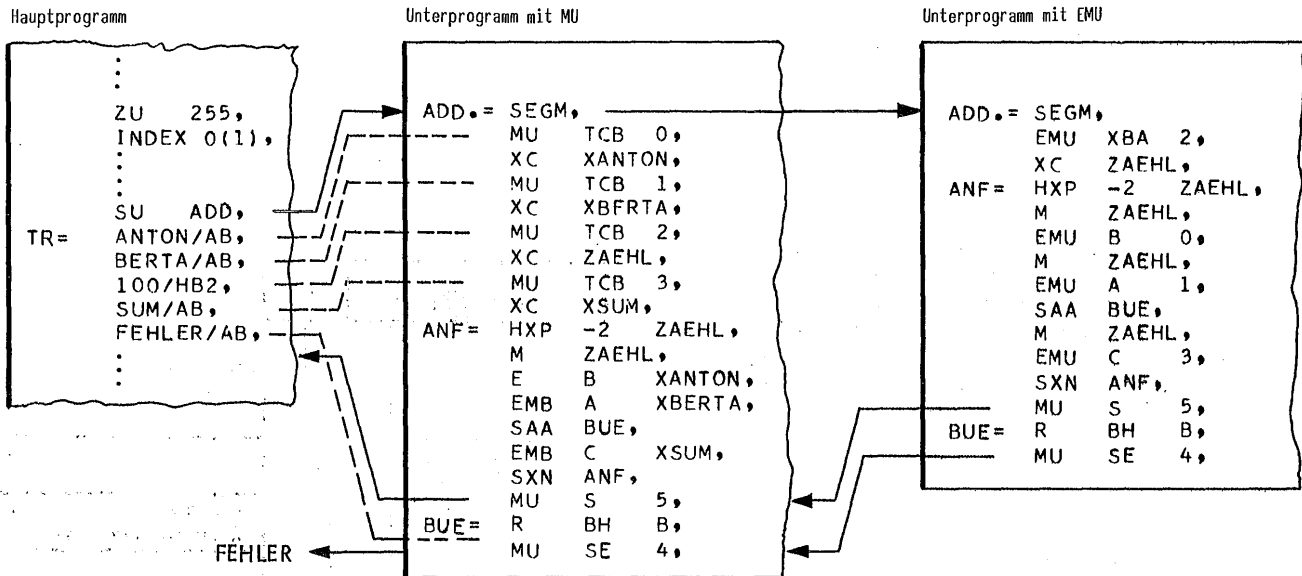


Bild 3.16a Versorgungsblock hinter SU

Unterprogrammname:	ADD	Addition
Aufgabe:	$a_x + b_x = c_x$, $x = n-2, n-4, \dots, 6, 4, 2, 0$	
Aufruf:	SU ADD,	
Versorgungsblock:	a/A, a = Anfangsadresse erster Summanden b/A, b = Anfangsadresse zweiter Summanden n/H, n = Anzahl der Durchläufe mal zwei c/A, c = Anfangsadresse der Summen f/A, f = Rücksprungadresse bei Bereichsüberschreitung	
Rückkehr:	hinter Versorgungsblock: Adresse des Befehls SU + 6	
Fehlerausgang:	wenn $c_x \geq 2^{16}$: Rücksprung nach f	
Speicherbedarf: (Halbwörter)	Parameterübernahme mit MU bzw. EMU	
	K-Bereich:	-- --
	V-Bereich:	-- --
	B-Bereich:	18 14
	D-Bereich:	-- --
	Indezellen:	4 1

3.16b Angaben zu den Unterprogrammen aus Bild 3.16a

Das Beispiel in Bild 3.16a ist eine Abwandlung des Beispiels in Bild 2.8a. Im ersten Unterprogramm wird mit Hilfe des Befehls MU die Adresse des jeweiligen Versorgungsparameters zum Adressenteil des Zweitbefehls TCB gemacht, und damit werden die Versorgungsparameter (ANTON, BERTA, 100, SUM) ins Register B gebracht. Anschließend werden sie mit XC in Indexzellen sichergestellt und in der Additionsschleife von dort jeweils geholt.

Im zweiten Beispiel wird mit dem Befehl EMU gearbeitet. Die Parameter ANTON, BERTA und SUM brauchen hier nicht vorher in Indexzellen abgelegt zu werden. In der

Additionsschleife werden sie nach der Ersetzung über EMU direkt aus dem Hauptprogramm geholt. Sie werden zum Adressenteil der Bringe-, Additions- und Speicherbefehle. Nur die für die Additionsschleife erforderliche Zählgröße 100 wird zweckmäßigerweise in einer Indexzelle abgelegt, nachdem sie zuvor über die Ersetzung mit EMU durch XBA ins Register B gebracht wurde.

Die Bilder 3.17 und 3.18 zeigen eine Zusammenstellung der Befehle MU und EMU, für den Fall, daß die Versorgungsparameter hinter dem Unterprogrammssprung stehen. (Das Entsprechende gilt, wenn sie vor dem Befehl SU stehen.)

Hauptprogramm	Unterprogramm mit MU		Unterprogramm mit EMU	
	Befehl ergibt Befehl		Befehl ergibt Befehl	
: SU unterprogrammname, tr+0 = parameter 1, tr+1 = parameter 2, : : tr+(x-1) = parameter x, tr+x = nächster befehl, : :	MU c 0 MU c 1 : : MU c (x-1) MU S x	c tr+0 c tr+1 : : c tr+(x-1) S tr+x	EMU c 0 EMU c 1 : : EMU c (x-1) MU S x	c parameter 1 c parameter 2 : : c parameter x S tr+x

tr = technische Rücksprungadresse
 c = Zweitcode
 x = Anzahl der Parameter

Bild 3.17 Allgemeiner Vergleich der Befehle MU und EMU

Hauptprogramm	Unterprogramm mit MU		Unterprogramm mit EMU	
	Befehl ergibt Befehl		Befehl ergibt Befehl	
: SU ADD, TR = ANTON/AB, BERTA/AB, 100/HB2, SUM/AB, FEHLER/AB, : :	MU TCB 0 MU TCB 1 MU TCB 2 MU TCB 3 MU SE 4 MU S 5	TCB TR TCB TR+1 TCB TR+2 TCB TR+3 SE TR+4 S TR+5	EMU B 0 EMU A 1 EMU XBA 2 EMU C 3 MU SE 4 MU S 5	B ANTON A BERTA XBA 100 C SUM SE TR+4 S TR+5

Bild 3.18 Vergleich der Befehle MU und EMU für das Beispiel in Bild 3.16a

D

TAS - Handbuch

Dez. 68

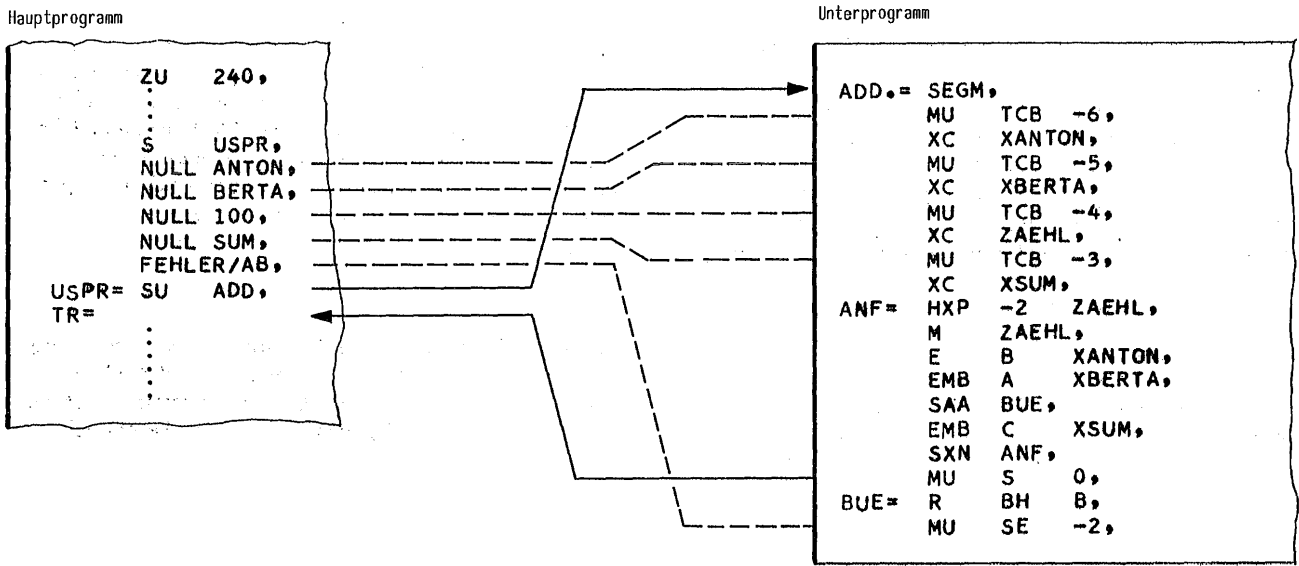


Bild 3.19 Versorgungsparameter vor dem Befehl SU

Im Bild 3.19 ist das Beispiel aus Bild 3.16a gezeigt, wenn die Versorgungsparameter vor dem Befehl SU stehen (bis 126 Parameter sind möglich). Solange sie nicht länger als 16 Bits sind, können sie im Adressenteil des Befehls NULL stehen.

Umfassen die Versorgungsparameter mehr als 16 Bits, so ist zusätzlich zu beachten, daß sie beim sequentiellen Ablauf des Programms nicht mit durchlaufen werden. Zu diesem Zweck wird der Versorgungsblock mit einem Sprungbefehl, der auf den Befehl SU führt, übersprungen.

Der Rücksprung vom Unterprogramm ins Hauptprogramm erfolgt auf die technische Rücksprungadresse, also direkt hinter SU.

3.3. Versorgungsbefehle

Der Versorgungsblock kann auch Versorgungsbefehle enthalten, die mit Hilfe des Befehls T zur Ausführung kommen. Das Bild 3.20 zeigt dies in einer Abwandlung des Beispiels aus Bild 3.16a.

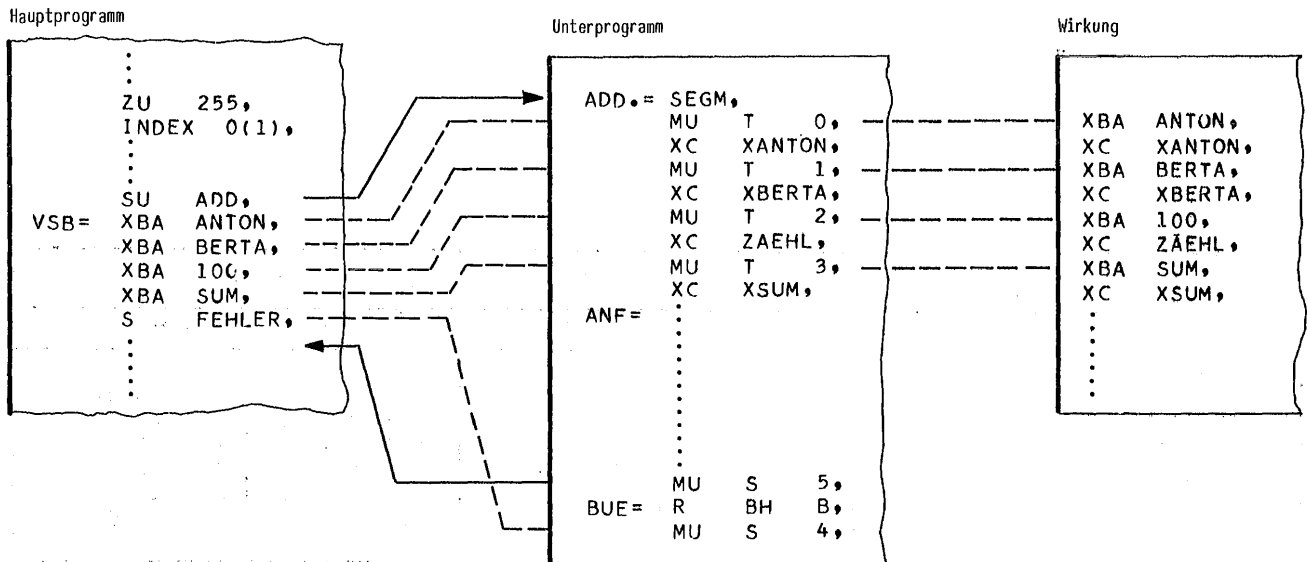


Bild 3.20 Versorgungsbefehle im Versorgungsblock hinter SU

Es ist günstig, wenn der Versorgungsblock direkt hinter dem Unterprogrammprung (SU) liegt, da die Versorgungsbefehle mit MU einfach erreicht werden können. Durch den Befehl T werden sie dann im Unterprogramm ausgeführt. Wird mit 24-Bit-Adressen gerechnet, so kann nicht mehr mit XBA gearbeitet werden, sondern es ist der Befehl TCB anzuwenden. Der erste Versorgungsbefehl in Bild 3.20 würde dann sein

TCB (ANTON/A),

Steht der Versorgungsblock vor dem Befehl SU und enthält er Sprungbefehle oder Befehle, welche wichtige Registerstände oder Speicherzellen verändern, so muß er mit einem Sprungbefehl, der auf SU führt, übersprungen werden. (Im übrigen gilt das in Abschnitt 2.5. Beschriebene.)

3.4. Adresse des Versorgungsblocks hinter SU

Es gilt das gleiche wie bereits im Abschnitt 2.3. geschriebene. In Bild 3.21 ist das Beispiel aus Bild 2.11 entsprechend abgeändert. Der Rücksprung erfolgt mit dem Befehl MU S 1.

Da hinter SU die Adresse des Versorgungsblocks steht, eignet sich der Ersetzbefehl EMU besonders dazu, die Parameter des Versorgungsblocks zu erreichen. Er bewirkt, daß die Adresse VSB zum Adressenteil des Befehls TCB wird. Durch Modifizierung dieses Befehls über MA werden die Daten des Versorgungsblocks zunächst ins Register B gebracht und von dort in einer Indexzelle abgelegt.

Im Versorgungsblock in Bild 3.21 fehlt die Adresse für einen Fehlerausgang (FEHLER). Die Versorgungsparameter können in diesem Beispiel nur über EMU erreicht werden. Der Rücksprung ins Hauptprogramm auf eine Adresse FEHLER wäre über EMU aber nicht möglich, da bei diesem Befehl das Register U nicht zurückgesetzt wird. (Es darf nur über den Befehl MU ins Hauptprogramm zurückgekehrt werden.) Soll vom Unterprogramm aus ein Fehlerprogramm im Hauptprogramm angesprungen werden, so muß die Fehleradresse wie in den Abschnitten 3.2. und 3.3. über den Befehl MU erreichbar sein.

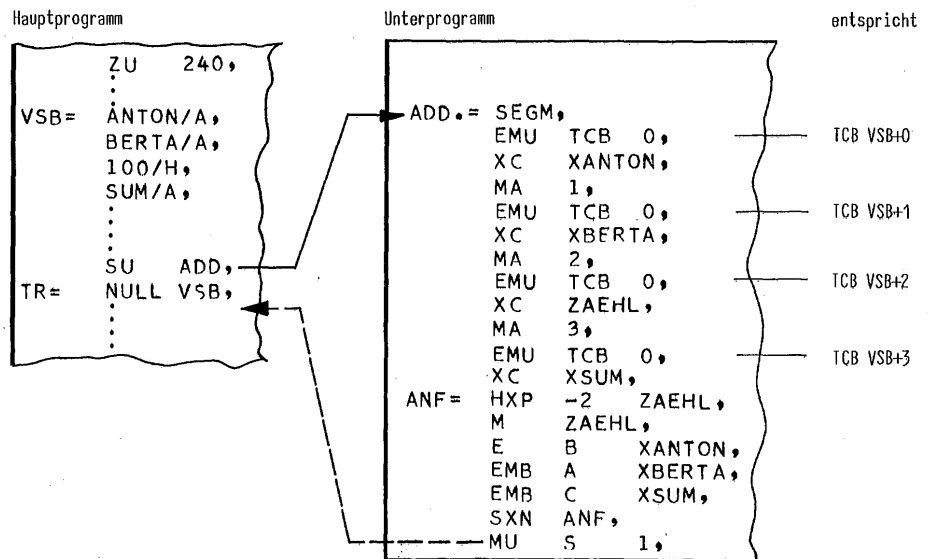


Bild 3.21 Beispiel Adresse des Versorgungsblocks hinter SU

3.5. Adresse des Versorgungsblocks im Register

Es gilt das in Abschnitt 2.2. allgemein über den Versorgungsblock geschriebene.

In Bild 3.22 ist das Beispiel in Bild 2.8a entsprechend für den Befehl SU abgeändert. Der Versorgungsblock enthält vier Parameter. Seine Adresse wird im Hauptprogramm ins Register A gebracht und steht dort dem Unterprogramm zur Verfügung.

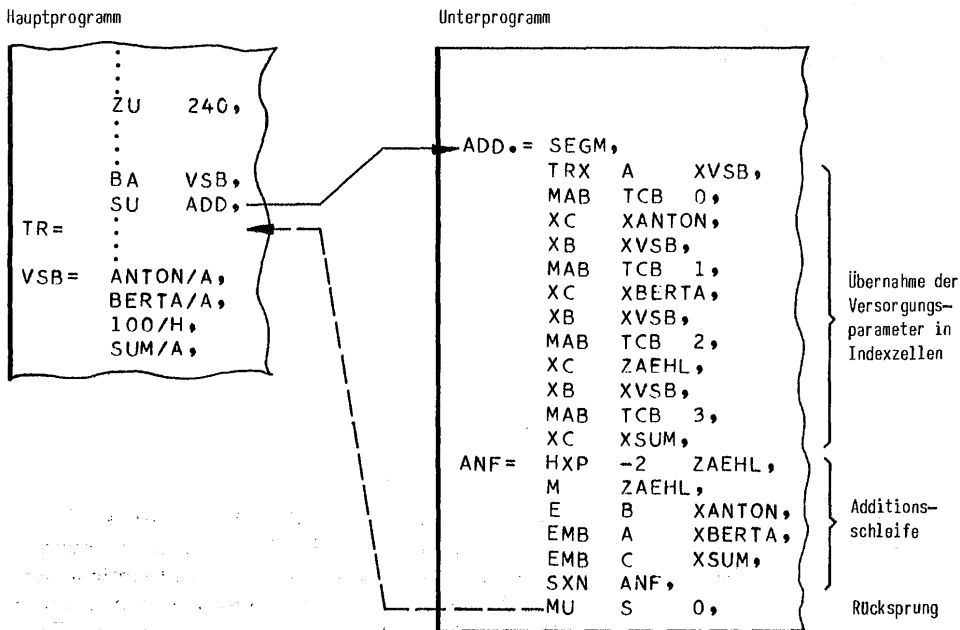


Bild 3.22 Beispiel für Adresse des Versorgungsblocks im Register

Die Versorgungsparameter wurden in Form von Konstanten angegeben und somit im K-Bereich abgelegt. Sollen die Parameter veränderlich sein, so ist jeder Konstanten noch V anzufügen. Alle Versorgungsparameter haben Halbwortlänge. In Bild 3.23 ist gezeigt, wie der Versorgungsblock aus Bild 3.22 auch als Literal des Befehls BA geschrieben werden kann.

Soll der Versorgungsblock im B- oder D-Bereich liegen, so kann das Register A nicht mit dem Befehl BA vorbesetzt werden. In diesem Fall wird mit dem Befehl B2 gearbeitet. Im Bild 3.24 ist das Beispiel aus Bild 3.22 abgewandelt, indem der Versorgungsblock in den D-Bereich gelegt wurde.

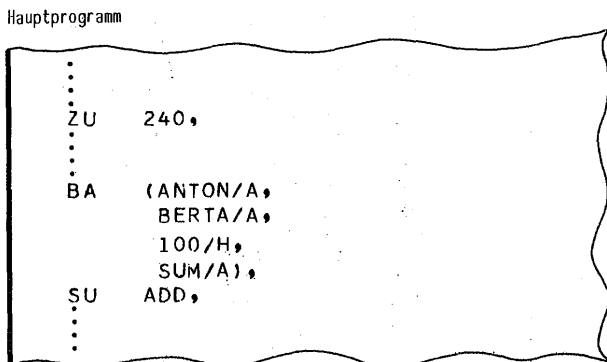


Bild 3.23 Versorgungsblock im Literal

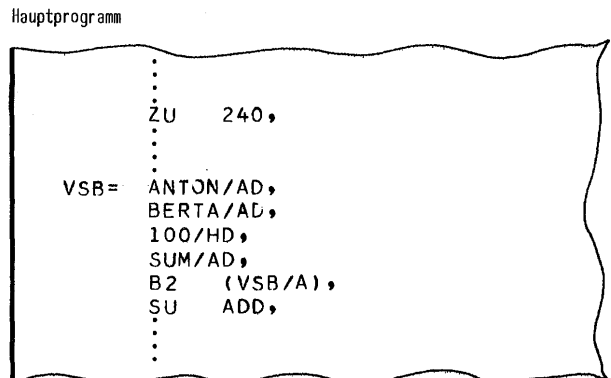


Bild 3.24 Versorgungsblock im D-Bereich

Im Unterprogramm Bild 3.22 wurde zuerst die Adresse des Versorgungsblocks aus dem Register A in die Indexzelle XVSB gebracht.

Danach wurden die Versorgungsparameter in Indexzellen übernommen. Dazu dient die Befehlsfolge

$$\begin{matrix} \text{XB} & \text{XVSB}, \\ \text{MAB} & \text{TCB } p, \\ \text{XC} & i, \end{matrix} \left. \vphantom{\begin{matrix} \text{XB} \\ \text{MAB} \\ \text{XC} \end{matrix}} \right\} \text{ ergibt: } \left\{ \begin{matrix} \text{TCB} & \text{VSB}+p, \\ \text{XC} & i, \end{matrix} \right.$$

Dies ist dann von Vorteil, wenn die Versorgungsparameter innerhalb einer Schleife oft benötigt werden. Es ist jedoch auch möglich, die Versorgungsparameter nicht in Indexspeicherzellen zu übernehmen, sondern direkt aus dem Versorgungsblock.

Der Versorgungsblock in Bild 3.22 enthält keine Fehleradresse (FEHLER), da sie in diesem Beispiel nicht über den Befehl MU erreicht werden könnte. Soll vom Unterprogramm aus ein Fehlerprogramm im Hauptprogramm angesprungen werden, so muß die Fehleradresse wie in den Abschnitten 3.2. und 3.3. über den Befehl MU erreichbar sein (siehe auch Abschnitt 3.4.).

3.6. Versorgungsblock in Indexzellen

Es gilt das in Abschnitt 2.6. geschriebene. Im Beispiel in Bild 3.25a erfolgt der Sprung ins Unterprogramm mit dem Befehl SU. Über den Befehl MU wird ins Hauptprogramm zurückgesprungen.

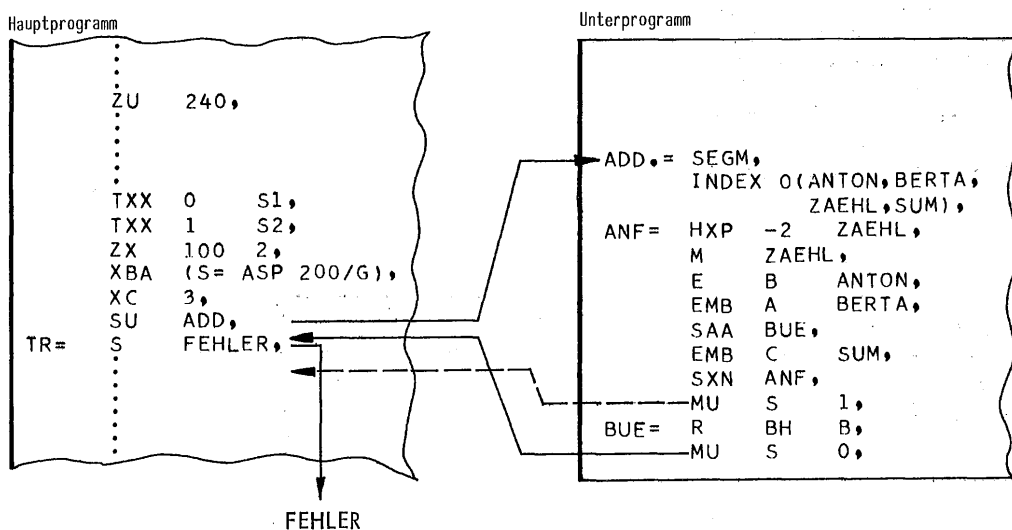


Bild 3.25 a Übergabe der Versorgungsparameter in Indexzellen

Unterprogrammname:	ADD	Addition
Aufgabe:	$a_x + b_x = c_x$, $x = n-2, n-4, \dots, 6, 4, 2, 0$	
Aufruf:	SU ADD,	
Versorgung:	Indexzelle 0: Anfangsadresse erster Summanden Indexzelle 1: Anfangsadresse zweiter Summanden Indexzelle 2: Anzahl der Durchläufe mal zwei Indexzelle 3: Anfangsadresse der Summen	
Rückkehr:	hinter technische Rücksprungadresse: Adresse des Befehls SU + 2	
Fehlerausgang:	wenn $c_x \geq 2^{48}$: Rücksprung auf Befehl SU+1 (von dort Sprung nach FEHLER)	
Speicherbedarf: (Halbwörter)	K-Bereich:	--
	V-Bereich:	--
	B-Bereich:	10
	D-Bereich:	--
	Indexzellen:	4

Bild 3.25 b Angaben zum Unterprogramm aus Bild 3.25 a

3.7. Umschalten der Indexbasis

Bei besonders umfangreichen Unterprogrammen ist es manchmal erforderlich, daß das Unterprogramm einen eigenen Indexbereich bekommt. Zu diesem Zweck wird das Indexbasisregister mit dem Befehl BCI oder ZI auf einen neuen Wert gesetzt, und damit wird ein neuer Indexbereich eröffnet. (Im Abschnitt "Befehle und Adressierung", 5.3. ist die Umschaltung der Indexbasis im einzelnen beschrieben.)

Die Indexbasis kann sowohl im Hauptprogramm als auch im Unterprogramm umgeschaltet werden.

Das Beispiel in Bild 3.26 zeigt die Umschaltung der Indexbasis im Unterprogramm. Mit dem Pseudobefehl ASP werden im Unterprogramm eine Anzahl Halbwörter freigehalten. Über den Befehl EMU wird jeder Versorgungsparameter aus dem Hauptprogramm zunächst ins Register B gebracht und dann mit TBC in die mit ASP reservierten Halbwörter im Unterprogramm abgelegt. Die Anfangsadresse dieses Speicherbereichs wird mit dem Befehl BCI zur neuen Indexbasis erklärt. Die Versorgungsparameter liegen somit in den vier ersten Zellen des neuen Indexbereichs. Vor dem Rücksprung ins Hauptprogramm muß die Indexbasis zurückgeschaltet werden. Dadurch ist auch der ursprüngliche Inhalt des Registers U mit der sichergestellten technischen Rücksprungadresse wieder eingestellt.

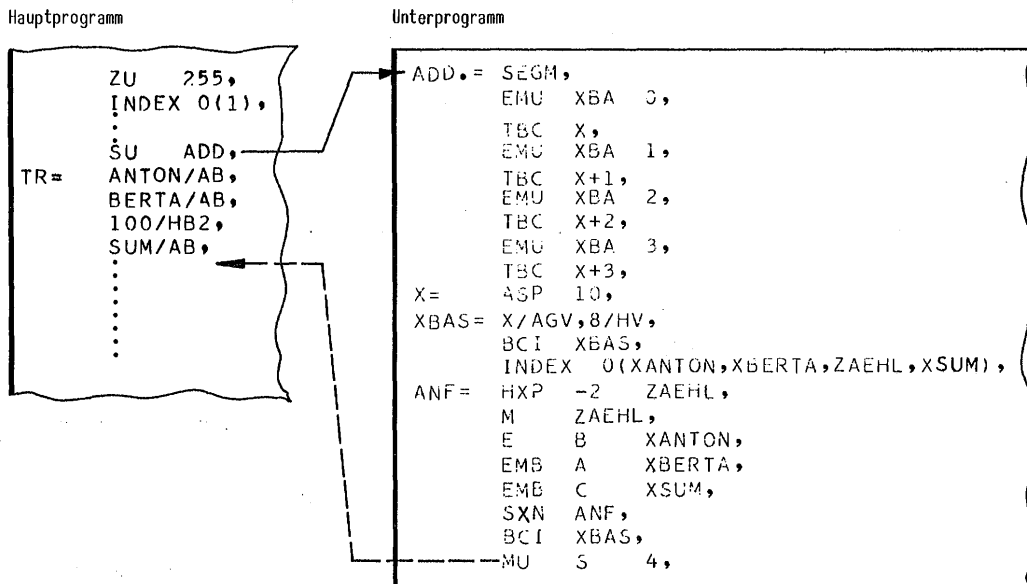


Bild 3.26 Umschalten der Indexbasis im Unterprogramm

In Bild 3.27 ist die Umschaltung der Indexbasis im Hauptprogramm mit dem Befehl BCI durchgeführt. Die vier Versorgungsparameter sind mit der Spezifikation V versehen, da der neue Indexbereich im V-Bereich liegen soll. Dahinter wird mit ASP ein Halbwort freigehalten für die spätere Aufnahme der technischen Rücksprungadresse. Die

Speicherzelle XBAS ist im linken Halbwort mit der Anfangsadresse des Versorgungsblocks belegt welche zur neuen Indexbasis wird. Im rechten Halbwort steht die Adresse der Indexzelle, auf die das Register U neu gesetzt werden soll. Der Befehl BCI enthält im Adressenteil die Adresse XBAS und bringt dadurch als neue Indexbasis

Hauptprogramm

```
ZU 240,  
.....  
XBAS= VERS/AGV,3/HV,  
VERS= ANTON/AV,  
BERTA/AV,  
100/HV,  
SUM/AV,  
TRU = ASP 1,  
BCI XBAS,  
SU ADD,  
BCI XBAS,  
.....
```

Index-
bereich
für Unter-
programm

Unterprogramm

```
ADD.= SEGM,  
INDEX 0(XANTON,XBERTA,  
ZAEHL,XSUM),  
ANF= HXP -2 ZAEHL,  
M ZAEHL,  
E B XANTON,  
EMB A XBERTA,  
EMB C XSUM,  
SXN ANF,  
MU S 0,
```

Bild 3.27 Umschalten der Indexbasis im Hauptprogramm

die Adresse des Versorgungsblocks ins Indexbasisregister. Somit sind die Versorgungsparameter die ersten Indexzellen des neuen Indexbereichs. Gleichzeitig wird das Register U auf 3 gesetzt, das ist die Indexadresse des letzten Versorgungsparameters (SUM). Beim Sprung ins Unterprogramm mit dem Befehl SU wird die technische Rücksprungadresse in die nächst höhere Indexzelle abgelegt. Das ist das mit ASP freigehaltene Halbwort hinter den

Versorgungsparametern. (Somit kann zu Beginn des Programms das Setzen des Registers U durch den Befehl ZU entfallen.)

Nach der Rückkehr aus dem Unterprogramm ins Hauptprogramm muß die Indexbasis auf ihren ursprünglichen Wert zurückgesetzt werden.

D

4. ERSETZUNGSTECHNIK

Beim TAS-Assembler ist ein Ersetzungsmechanismus vorhanden. Er erlaubt es, beliebige Ausschnitte eines Programmes, die sich wiederholen, nur einmal niederzuschreiben.

Zu diesem Zweck muß der sich wiederholende Ausschnitt festgelegt (definiert) werden. Dies wird Ersetzungsdefinition genannt. Der Aufbau ist im Kapitel 4.1. erläutert.

Im anschließenden Programmteil braucht nur noch auf den durch die Ersetzungsdefinition festgelegten Programmteil Bezug genommen zu werden. Anstelle dieses Ersetzungsbezugs wird durch den Assembler der mit der Ersetzungsdefinition festgelegte Ersetzungstext fest eingesetzt. Der Ersetzungsbezug ist im Kapitel 4.2. näher erläutert.

Ersetzungen können auch verschachtelt auftreten, d.h. in einer Ersetzungsdefinition kann wiederum ein Ersetzungsbezug vorhanden sein. Es ist eine mehrfache Verschachtelung erlaubt (siehe Kapitel 4.4.).

Im Kapitel 4.5. ist dann die Möglichkeit aufgezeichnet, wie mit Hilfe von Parametern die Ersetzungstexte flexibel gemacht werden können.

Wird die Ersetzungstechnik in einem Programm benutzt, so muß im UEBERSETZE-Kommando als Spezifikation für die Sprache TASE angegeben werden.

4.1. Ersetzungsdefinition

Bevor in einem Programm auf einen Ersetzungstext hingewiesen (Bezug genommen) werden kann, muß er erst festgelegt (definiert) werden. Eine Ersetzungsdefinition hat die Form

```
*(name = x),      name: Name des Ersetzungstextes
                   x: Ersetzungstext, beliebiger
                   Programmausschnitt
```

Der Stern mit der anschließenden Klammer leitet die Ersetzungsdefinition ein. Anschließend muß ein Name angegeben werden. Er muß die für Namen vorgeschriebene Form haben. Unter diesem Namen wird später der Ersetzungstext aufgerufen. Es können verschiedene Ersetzungstexte unter einem Namen geführt werden. Es gilt dann immer der - in der Reihenfolge der Niederschrift - zuletzt definierte Ersetzungstext.

Nach einem Gleichheitszeichen (Zuordnungszeichen) folgt der Ersetzungstext, das ist der Text, der wiederholt vorkommt. Er kann ein beliebiger Ausschnitt eines Programmes sein und kann im Minimalfall aus einem

einigen Zeichen, aber auch aus einer Folge von Konstanten oder Befehlen bestehen. Der Ersetzungstext muß jedoch nach dem Einsetzen in die gewünschte Stelle eines Programms mit seiner Umgebung einen erlaubten Zusammenhang haben.

Abgeschlossen wird die Ersetzungsdefinition durch eine Klammer. Wie bei Befehlen oder Konstanten bildet die Ersetzungsdefinition eine Informationseinheit und muß durch ein Komma oder einen Kommentar abgeschlossen werden.

```
*(PI= 3.1415926),
*(A= EMIL - KARL + 2),
*(SH= SH AL 8,
      AA 170,
      SH ZL 24,)
```

Bild 4.1 Beispiele für Definitionen

4.2. Ersetzungsbezug

Ist ein Ersetzungstext definiert worden, so kann er an jeder Stelle im Programm mit seinem Namen aufgerufen werden. Der Ersetzungsbezug hat die Form

```
*(name)           name: Name, unter dem der Ersetzungstext
                   definiert wurde
```

Anstelle dieses Ersetzungsbezugs wird beim Assemblieren durch den TAS-Assembler der Ersetzungstext eingesetzt, der mit dem Namen (als letzter) definiert worden ist.

Der Ersetzungsbezug kann an beliebiger Stelle innerhalb des Programms stehen, jedoch immer erst nach der Ersetzungsdefinition.

```
ZX *(ANZAHL),
ANF= *(SH),
B ANTON + *(CH),
SH *(A) 8,
```

Bild 4.2 Beispiele für Ersetzungsbezüge

4.3. Einfache Ersetzungen

Bei der Programmierschrift muß erst die Ersetzungsdefinition angegeben werden und danach können die Ersetzungsbezüge aufgeführt werden. Eine Ersetzung hat die im Bild 4.3 angegebene Form. Der Ersetzungstext muß sich nach dem Assemblieren sinnvoll in das Programm einpassen.

Niederschrift

```

:
: *(name = texte), ----- Definition
:
: ... *(name)....., ----- Bezug
:
: *(name)...., ----- Bezug
:

```

nach Assemblierung

```

:
: .....texte.....,
:
: texte.....,
:

```

Bild 4.3 Allgemeine Form der Ersetzung (vor und nach der Assemblierung)

Niederschrift

```

:
: *(ANZ= 12),
:
: ZX *(ANZ) XA,
:
: B *(ANZ),
:
: AA *(ANZ),
:
: *(ANZ),
:

```

nach Assemblierung

```

:
: ZX 12 XA,
:
: B 12,
:
: AA 12,
:
: 12,
:

```

Bild 4.4 Beispiel einer einfachen Ersetzung (vor und nach der Assemblierung)

Unter einem Ersetzungsamen, der schon einmal zur Definition eines Ersetzungstextes verwendet wurde, kann jederzeit ein neuer Text definiert werden. Die ältere Ersetzungsdefinition ist damit gelöscht.

4.4. Verschachtelte Ersetzung

In einem Ersetzungstext können wieder neue Ersetzungsdefinitionen durchgeführt werden oder weitere Ersetzungsbezüge enthalten sein, so daß eine mehrfach verschachtelte Ersetzung möglich ist. Die Ersetzung erfolgt aber auch hier nur dann, wenn die ineinandergeschachtelten Ersetzungstexte zuvor mit einem Ersetzungsamen versehen wurden. Es können beliebig viele Ersetzungsverachtelungen, Aufrufe und Definitionen durchgeführt werden.

4.5. Ersetzungsparameter

Mit Hilfe der Ersetzungsparameter ist es möglich, die Ersetzungen noch flexibler zu gestalten.

Bei der Ersetzungsdefinition kann innerhalb des Ersetzungstextes ein Bezug auf einen Ersetzungsparameter angegeben werden. Dieser Parameterbezug hat die Form

*(z) z: Ziffer 1,2,...

Die Ziffer innerhalb der Klammer bezieht sich auf einen Parameter, der beim Ersetzungsbezug angegeben ist.

Niederschrift

```

:
: *(ANZAHL= 12),
:
: ZX *(ANZAHL) XA,
:
: *(ANZAHL= 4),
:
: ZX *(ANZAHL) XA,
:

```

nach Assemblierung

```

:
: ZX 12 XA,
:
: ZX 4 XA,
:

```

Bild 4.5 Ersetzungsdefinitionen mit gleichem Namen (vor und nach Assemblierung)

Niederschrift

```

    :
    :
    *(A2= 7),
    *(A3= 4711),
    :
    *(ANZAHL= ZX *(A2) XA,
    B *(A3)),
    :
    *(ANZAHL),
    :
    *(A2= 113),
    *(A3= BERLIN),
    :
    *(ANZAHL),
    :
    :
  
```

nach Assemblierung

```

    :
    :
    ZX 7 XA,
    B 4711,
    :
    :
    ZX 113 XA,
    B BERLIN,
    :
    :
  
```

Bild 4.6 Beispiel für verschachtelte Ersetzung (vor und nach Assemblierung)

Niederschrift

```

    :
    :
    *(ANZAHL= ZX *(1) XA,
    B *(2)),
    :
    *(ANZAHL,7,4711),
    :
    *(ANZAHL,113,BERLIN),
    :
    :
  
```

name
 p₁ für *(1)
 p₂ für *(2)

nach Assemblierung

```

    :
    :
    ZX 7 XA,
    B 4711,
    :
    ZX 113 XA,
    B BERLIN,
    :
    :
  
```

- *(1) wird ersetzt durch 7 bzw. 113
- *(2) wird ersetzt durch 4711 bzw. BERLIN

Bild 4.7 Beispiel für Ersetzungsparameter (vor und nach Assemblierung)

Beim Ersetzungsbezug werden hinter dem Ersetzungsnamen - durch Komma getrennt - die Ersetzungsparameter angegeben. Der Ersetzungsbezug hat dann die Form

* (name, p₁, p₂, ...)

name: Ersetzungsname
 p_x: Parameter x

Beim Assemblieren wird der Parameterbezug durch den Parameter ersetzt. Es gilt

- * (1) wird ersetzt durch p₁
- * (2) wird ersetzt durch p₂
- usw.

Damit ist es möglich, einen sich wiederholenden Programmausschnitt in einigen Teilen zu variieren - mit anderen Worten - Programmausschnitte, die sich nur geringfügig unterscheiden, nur einmal als Ersetzungstext zu definieren und durch Parameterangabe die Unterschiede zu berücksichtigen.

Beim Ersetzungsbezug brauchen nicht alle Parameter angegeben zu werden. Die nicht benötigten Parameter werden fortgelassen. Die Tatsache, daß ein Parameter fortgelassen wurde, muß durch Setzen eines Kommas zum Ausdruck gebracht werden (leerer Parameter). Sollen zum Beispiel die Parameter p₂, p₄ und p₆ fortgelassen werden, so ist zu schreiben

* (name, p₁ ,, p₃ ,, , p₅)

Niederschrift

```
⋮
*(ANZAHL=
  *(1) B      *(2),
        ML*(3) *(4)),
⋮
*(ANZAHL,ANFANG=,ANTON,R,4711),
⋮
*(ANZAHL,,BERTA,,BERLIN),
⋮
```

name
p₁ = leer
p₂ für *(2)
p₃ = leer
p₄ für *(4)

nach Assemblierung

```
⋮
ANFANG= B ANTON,
        MLR 4711,
⋮
B BERTA,
ML BERLIN,
⋮
```

Bild 4.8 Beispiel für leere Parameter (vor und nach Assemblierung)

Niederschrift

```
⋮
*(FAKT3= 0.9785),
⋮
*(DIV= 23120),
⋮
*(V15=
  *(1)= BQ      *(2),
        BT      *(3),
        RT      AH,
        BT      *(4),
        R       *(5) H,
        CT      *(3) + 2,
        ML*(6) (*(FAKT3)),
        AA      *(DIV),
⋮
        MU      S 0),
⋮
*(V15,STRING,'FFF00',ST13,ST14,SB,R),
⋮
```

- 1. Ersetzungsdefinition
- 2. Ersetzungsdefinition
- 3. Ersetzungsdefinition

nach Assemblierung

```
⋮
STRING= BQ ('FFF00'),
        BT ST13,
        RT AH,
        BT ST14,
        R  SB H,
        CT ST13 + 2,
        MLR (0.9785),
⋮
AA 23120,
⋮
MU S 0,
⋮
```

name
Parameter p₁ bis p₆ für *(1) bis *(6)
Literale siehe Kapitel 5.

Bild 4.9 Beispiel mit Verschachtelung und Parameter (vor und nach Assemblierung)

7. Makrotechnik (Makrosprache)

Bei der Anwendung der Makrotechnik kann der Benutzer eine Erweiterung der TAS-Sprache erzielen, für den Programmierer kann dies ggf. auch eine Vereinfachung bedeuten.

Die Makrotechnik erfordert zusätzliche Pseudobefehle, die es ermöglichen, Folgen von Informationseinheiten als Makros zu definieren. Nach der Definition (Vereinbarung) können diese Folgen durch Aufrufe (Makroaufrufe) im Quellenprogramm eingesetzt werden. Die einzelnen Pseudobefehle der Makrotechnik ermöglichen es, Makros zu vereinbaren, aufzurufen und ggf. das Einsetzen der Makros zu steuern, um die generierte Folge dem Zweck des Aufrufs anzupassen.

Mit Hilfe von formalen Parametern läßt sich die Quelle vielseitiger gestalten. Alle durch die Vereinbarung anfallenden Informationseinheiten (Makrotext) werden vom Assembler zunächst überlesen bzw. abgespeichert und erst beim Aufruf interpretiert.

Der Benutzer kann in seiner Quelle z. B. auch Bedingungen (Versionen) stellen, von denen es abhängt, ob eine Folge von Informationseinheiten eingesetzt werden soll oder nicht. Desgleichen können in diesem Zusammenhang Makros z. B. als Teilprogramme definiert werden, die bei Bedarf abzurufen sind.

Durch einzelne im Makro enthaltene Pseudobefehle kann auch direkt auf die aktuellen Werte von formalen Parametern Bezug genommen werden.

Aktuelle und formale Parameter werden auch als Makrokonstante bzw. Makrovariable bezeichnet.

Je nach Zweck des Aufrufs steuert der Benutzer also über den formalen Parameter den Makrotext (bei entsprechendem Aufbau des Makros) mit Hilfe von aktuellen Parametern und evtl. zusätzlichen Pseudobefehlen.

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

7.1.1. Anwendung der Makrosprache

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

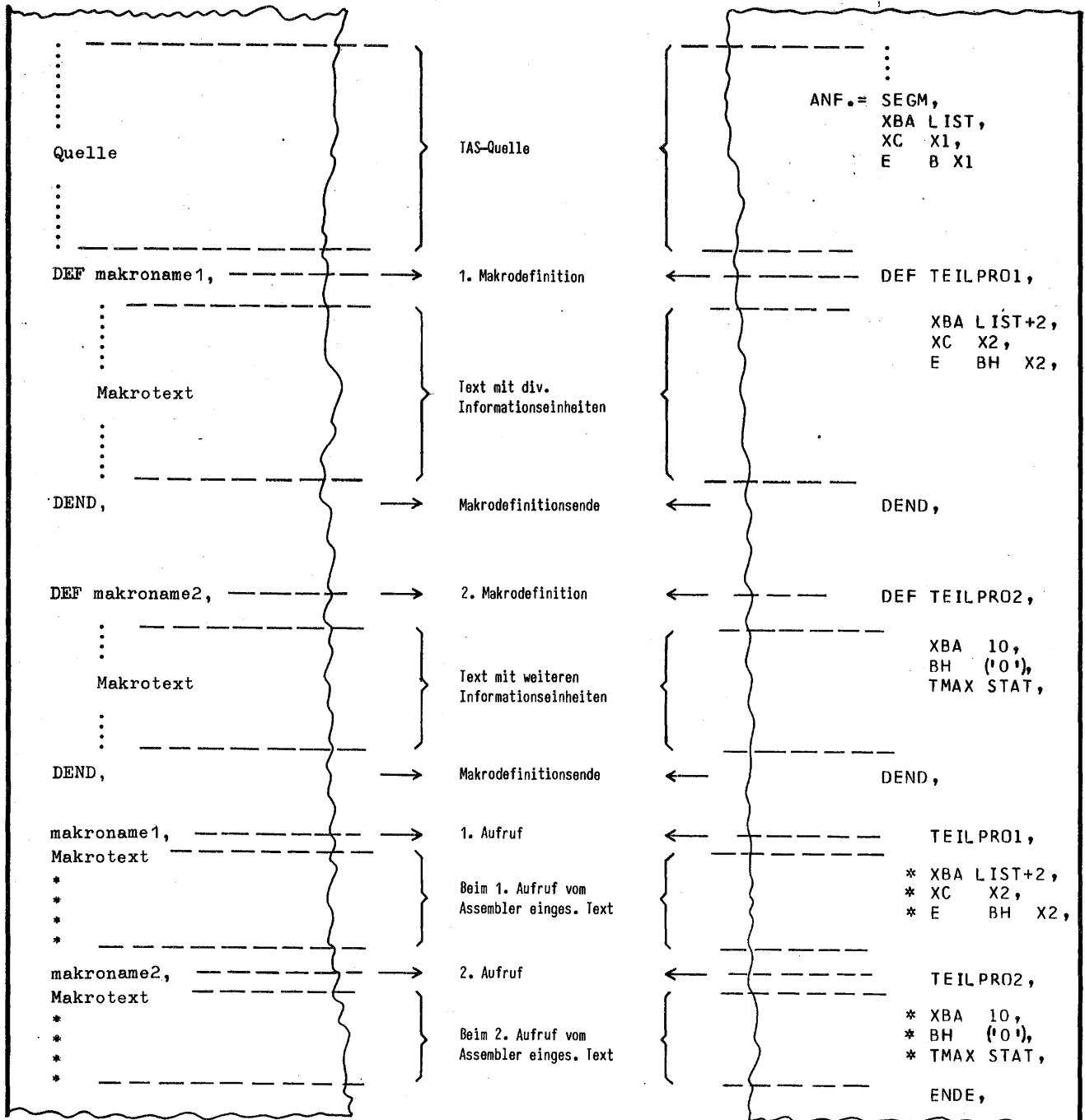
Makroaufrufe können jedoch außerdem zur Erweiterung des Befehlsumfanges verwendet werden; sie haben also eine doppelte Funktion:

Einerseits können Makroaufrufe als Anweisungen verstanden werden, häufig benötigte vorbereitete Programmteile einzusetzen. Andererseits ist es dem Benutzer eines ausgetesteten Makros im allgemeinen gleichgültig, wie die aufgrund des Makroaufrufs generierte Befehlsfolge aussieht, wenn er weiß, welche genau definierte Leistung diese Befehlsfolge erbringt.

Der Makroaufruf steht stellvertretend für diese Leistung und kann daher wie ein zusätzlicher Befehl oder Pseudobefehl verwendet werden (durch Anwendung des DRUCK-Befehls kann der Programmierer wählen, ob der aufgrund des Makroaufrufs generierte Quelltext im Übersetzerprotokoll erscheinen soll oder nicht; im letzteren Fall unterscheidet sich der Makroaufruf hinsichtlich der Protokollierung nicht von einem normalen Befehl). Mit Hilfe der Makrosprache können also zur Ergänzung des Befehlsumfanges neue Befehle mit genau definierten Leistungen eingeführt werden. Der Benutzer dieser Befehle braucht dabei die Makrosprache nicht zu kennen; die Kenntnis der Leistungen der neuen Befehle genügt.

7.1. Elemente

Ein Makro wird nur einmal vereinbart (Makrodefinition), mit einem Namen versehen (Makroname) und kann beliebig oft aufgerufen werden (Makroaufruf). Die Aufrufe sind also für den Assembler Anweisungen, vorher bereitgestellte Makrotexte einzusetzen. Wie noch aus den später aufgeführten Beispielen zu ersehen ist, können Makrotexte beim Einsetzen verändert werden. Einzelne Informationseinheiten können formale Parameter enthalten. Diesen werden beim Makroaufruf, d. h. während der Interpretation des Makros, aktuelle Werte, nämlich die aktuellen Parameter zugeordnet. Formale Parameter dürfen an beliebigen Stellen von Informationseinheiten stehen und werden unmittelbar bei der Interpretation der jeweiligen Informationseinheit durch ihren aktuellen Wert ersetzt.



Erläuterung:

Beachte! Vom Assembler eingesetzte Texte sind auch wie im Original mit * gekennzeichnet

Bild 7. 1 Allgemeine Form des einfachen Makros mit einem Beispiel

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

Weitere Pseudobefehle (WIED-Befehle) sind Anweisungen an den Assembler, bestimmte Folgen mehrfach zu interpretieren. In der Maschinensprache werden solche "Schleifen" im allgemeinen durch mehrere einzelne Befehle organisiert. Während die Maschinenbefehle im allgemeinen ihre Objekte (Konstanten, Befehlsfolgen etc.) adressieren, erfolgt in der Makrosprache die Bezugnahme mit Hilfe vereinbarter Namen. Die Makrosprache enthält daher Pseudobefehle, die bestimmte Objekte der Makrosprache (Parameter, Makros) vereinbaren und diese Objekte gleichzeitig mit Namen benennen, um einen Bezug auf diese Objekte möglich zu machen.

7.1.2. Vergleich der Makrosprache mit einer Maschinensprache

Die Interpretation der Makrosprache durch den Assembler hat große Ähnlichkeit mit der Interpretation einer Maschinensprache durch einen Rechner. Die Kenntnis des letzteren Interpretationsvorgangs erleichtert daher das Verständnis der Handhabung und der Anwendungsmöglichkeiten der Makrosprache. Für Programmierer, die diese Kenntnis besitzen, werden daher im folgenden die beiden Interpretationsvorgänge verglichen:

Sowohl Maschinenprogramme als auch Quellenprogramme bestehen aus einzelnen Informationseinheiten, die nacheinander interpretiert werden. Die Informationseinheiten der Maschinensprache sind Befehle. Die Informationseinheiten der Makrosprache sind einzelne, in der Quellsprache dargestellte Befehle, Konstanten und Pseudobefehle.

Diese Informationseinheiten werden bei der Bearbeitung eines Programms im allgemeinen der Reihe nach interpretiert. (Der Rechner interpretiert Befehle. Dementsprechend interpretiert der Assembler Pseudobefehle und in der Quellsprache dargestellte Befehle und Konstanten: Bei den Pseudobefehlen besteht die Interpretation darin, daß sie ausgeführt, bei Befehlen und Konstanten darin, daß sie in die Maschinensprache übersetzt werden.)

Sowohl in der Maschinensprache als auch in der Makrosprache gibt es jedoch Informationseinheiten, die eine Unterbrechung des kontinuierlichen Interpretationsvorgangs bewirken:

Bestimmte Maschinenbefehle (z.B. die Unterprogramm- (Sprünge) sind Anweisungen an den Rechner, die Ausführung einer Folge von Befehlen zu unterbrechen und als nächstes eine bestimmte andere Folge zu interpretieren. Dementsprechend sind bestimmte Pseudobefehle der Makrosprache (die sogenannten Makroaufrufe) Anweisungen an den Assembler, die Interpretation einer Folge von Informationseinheiten zu unterbrechen und als nächstes eine vorher vereinbarte Folge von Informationseinheiten zu interpretieren.

Andere Pseudobefehle der Makrosprache entsprechen den bedingten Sprungbefehlen der Maschinensprache. Sie sind Anweisungen an den Assembler, bestimmte Folgen von Informationseinheiten nur dann zu übersetzen, wenn bestimmte Bedingungen erfüllt sind (VERS-Befehle).

7.2. Einfache Makros

Ein Makro muß, bevor es aufgerufen wird, vereinbart sein. Diese Vereinbarung oder Definition eines Makros hat die Form

```
DEF name,      name: Name des Makros, unter dem es
                aufgerufen werden kann.

                text,      text: Makrotext, der anstelle des
                Aufrufs eingesetzt werden
                soll.

DEND,
```

Der Makrotext muß mit dem Pseudobefehl DEF eingeleitet und dem Pseudobefehl DEND abgeschlossen werden.

Der Name des Makros muß die für Namen erlaubte Form haben. Er darf jedoch nicht mit dem Namen eines Befehls, eines Pseudobefehls oder dem Namen eines Makros, das allgemeine Gültigkeit hat, übereinstimmen (z.B. R&LIES, R&DRUCKE bei Benutzung des TAS-Standardrahmens).

Nachdem das Makro vereinbart wurde, kann es durch seinen Namen aufgerufen werden:

```
name,      name: bei der Definition vereinbarter Name des Makros
```

In Bild 7.1 ist die allgemeine Form eines einfachen Makros mit einem Beispiel dargestellt. Die Makros mit den Namen TEILPRO1 und TEILPRO2 fungieren hier als eine Art von Teilprogrammen. In abgewandelter Form kann man mit dem Makro eine beliebig lange Folge von Informationseinheiten festlegen, ähnlich dem Prinzip der Unterprogramme und seiner Aufrufe, ohne daß im Fall des Makros vom Benutzer eine Rücksprungadresse festzulegen ist. Sowohl beim Makro als auch beim Unterprogramm sind z. B. die Befehlsfolgen vorgegeben. Im Gegensatz zur Unterprogrammtechnik jedoch erscheint im lauffähigen Operator das Makro so oft wie es bei der Übersetzung aufgerufen wird. Ein zusätzlicher Bedarf an Kernspeicherraum ist dadurch erforderlich.

Das Makro darf wieder Makro- und andere Definitionen enthalten, wie den späteren Beschreibungen zu entnehmen ist. Bei der Definition wird das Makro vom Assembler noch nicht bearbeitet, sondern zunächst aufbewahrt und erst beim Aufruf interpretiert (siehe auch Abschnitt 7.1.). Der Makroaufruf kann je nach Bedarf innerhalb der Quelle beliebig oft verwendet werden.



7.3. Makros mit einem Parameter

Bei der Definition eines Makros kann ein Teil (oder mehrere Teile) einer Informationseinheit als Variable in Form eines formalen Parameters so oft wie erforderlich im Makrotext angegeben werden. Dies ist mit Hilfe des formalen Parameters und des aktuellen Parameters möglich.

Vor dem Aufruf muß der formale Parameter im Quellprogramm in der folgenden allgemeinen Form niedergeschrieben sein:

+(name) name: Name des formalen Parameters

Der formale Parameter darf also im Makrotext überall dort stehen, wo erst beim Aufruf des Makros ein Wert eingesetzt werden soll. Die Möglichkeiten, Teile einer Informationseinheit durch einen formalen Parameter zu ersetzen und später dafür den aktuellen Parameter anzugeben, sind dadurch begrenzt, daß der aktuelle Parameter folgende Zeichen nicht enthalten darf:

- -- zwei aufeinanderfolgende Minuszeichen,
- , Komma,
- = Gleichheitszeichen,
- ␣ Space,
- () runde Klammern einzeln (paarweise erlaubt),
- ' Apostroph einzeln (paarweise erlaubt),
- '' Doppelapostroph (paarweise erlaubt).

In Kommentaren und Überschriften ist ein Parameter nicht erlaubt. Bei Konstanten muß darauf geachtet werden, daß die Zeichenfolge +(nicht als Text auftaucht, sondern nur zur Einleitung eines formalen Parameters dient.

Der aktuelle Parameter darf enthalten:

- die Buchstaben A bis Z,
- die Ziffern 0 bis 9,
- die Sonderzeichen + Plus,
- Minus,
. Punkt,
* Stern,
CR Wagenrücklauf
(Kartende),
| Senkrechtstrich,
() runde Klammern
(paarweise),
'' Apostroph paarweise.

Zwei Minuszeichen dürfen nicht aufeinanderfolgen.

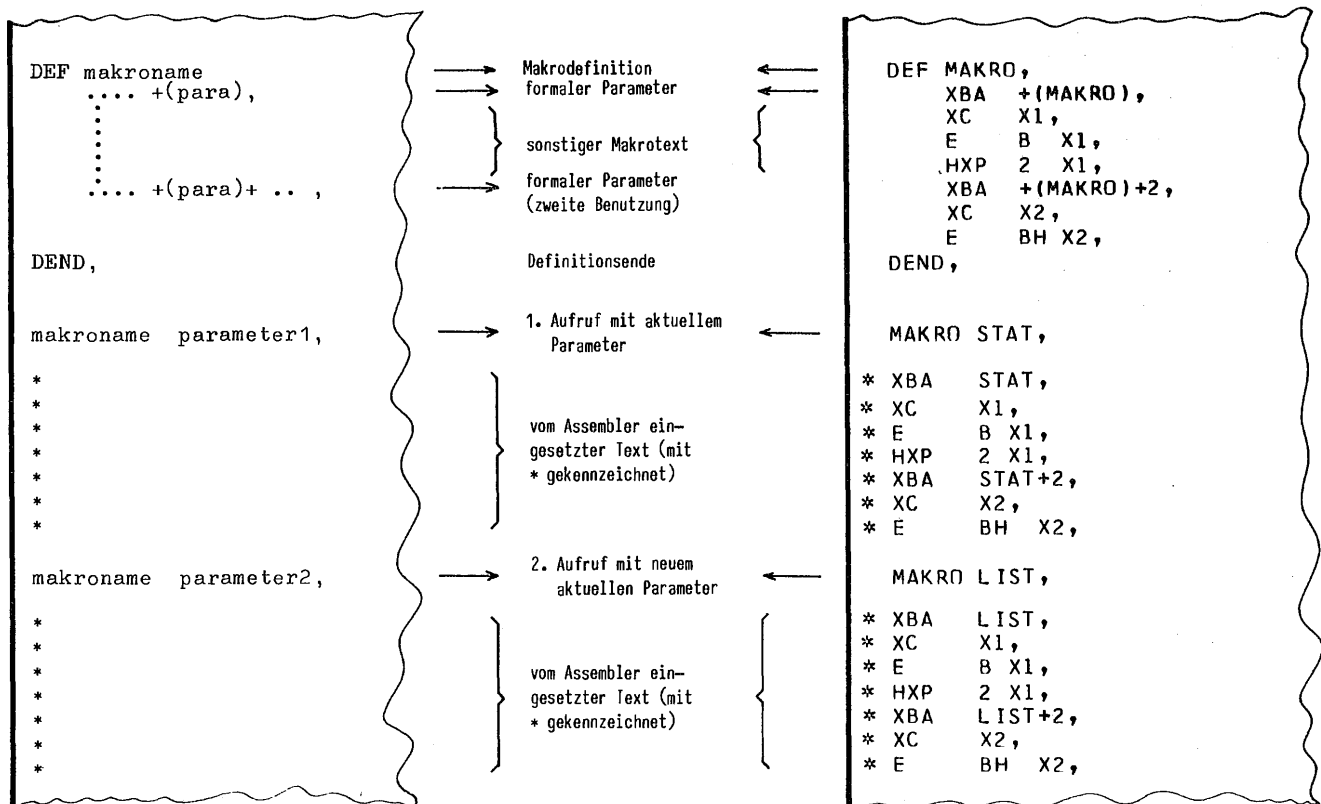


Bild 7.2 Allgemeine Form des Makros mit einem formalen Parameter und Beispiel
- Name des formalen Parameters und Makroname sind hier immer identisch.
- Aufrufe mit entsprechenden aktuellen Parametern sind mehrfach möglich -

In Bild 7.2 ist die allgemeine Form des Makros mit einem Parameter und Beispiel gezeigt. Wird nur ein Parameter verwendet, müssen die Namen des Makros und des formalen Parameters identisch sein. Beim Aufruf mit Angabe eines aktuellen Parameters wird dem formalen Parameter dieser Wert zugeordnet.

Makros mit mehreren Parametern sind in Abschnitt 7.4. beschrieben.

Der Makroaufruf mit einem Parameter hat die allgemeine Form:

makro para, makro: frei zu vereinbarenden Makro-
name
para: aktueller Parameter für den ange-
gebenen formalen Parameter

Weiterhin darf ein aktueller Parameter sein:

- eine Tetradenfolge (durch Apostroph (') eingeschlossen),
- eine Oktadenfolge (durch Doppelapostroph (' ') eingeschlossen),
- eine Textfolge (mit der Zeichenfolge '(eingeleitet und mit der Zeichenfolge ')' abgeschlossen),
- eine Liste von aktuellen Parametern, die in runde Klammern eingeschlossen sind.

Wie auch aus dem Beispiel ersichtlich, wird also nicht nur das Makro aufgerufen sondern auch dem formalen Parameter, der mehrmals im Makrotext angegeben werden kann, ein aktueller Wert beigegeben.

Werden nun bei einer Makrodefinition oder Wiederholungsanweisung mehrere Parameter benötigt, wird der Pseudobefehl FORM angewendet (s. Bild 7.3). Er verlangt, daß der implizit vorgegebene Parameter durch eine Parameterliste erweitert wird. Der implizite formale Parameter erhält den gleichen Namen wie den des Makros. Wie später aus den Beispielen zu ersehen ist, können mehrere Listen definiert werden.

Niederschrift

```
DEF ANTON,
  FORM ANTON (A,B,C),
  B   +(C),
  C   A+(A),
  BA  +(B),
  AA  +(A),
DEND,
```

Formatdeklaration
in einer Liste
formale Parameter aus
Liste als Adreßteile
eingesetzt

Aufruf

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

Makroaufruf mit Angabe
der aktuellen Parameter

Übersetzung

```
* B   (3),
* C   A1,
* BA  2,
* AA  15,
```

Bild 7.3 Formatdeklaration mit Parameterliste bei eingehaltener Reihenfolge der Zuordnung

Allgemeine Form:

FORM p (u₁, u₂, ..., u_n), p= Name des Parameters, der
aufgegliedert werden soll
u= Name für die Parameter
der Parameterliste

7.4. Makros mit mehreren Parametern (Parameterlisten)

Im Vergleich zu Abschnitt 7.3. besteht bei der Verwendung von mehreren Parametern die Möglichkeit, eine ganze Reihe von Informationseinheiten verschiedenartig zu vereinbaren. Ein einmal mit Makros bzw. Parametern festgelegtes Programm kann dadurch jederzeit variabel gestaltet werden.

FORM kann als Zuordnungsanweisung verstanden werden. Er bezieht sich mittels der formalen Parameter (-Listen) auf die in den aktuellen Parameter (-Listen) angegebenen Werte. Wenn auch der Name des Makros und der nach FORM folgende für die erste Parameterliste identisch sein müssen, so können doch aus der ersten Liste Verbindungen zu weiteren Listen hergestellt werden.

Allgemeine Form des Pseudobefehls FORM mit Unterlisten:

FORM p (u₁, u₂, ..., u_n), p= Name des Parameters, der aufgliedert werden soll
u= Name für die Parameterlisten

Die nach FORM p in der Liste aufgeführten Namen können mit weiteren FORM - Befehlen gesondert aufgliedert werden. Beispiel:

FORM p (u₁, u₂, ..., u_n),
FORM u₁ (a, b, ..., h),
:
:
FORM u_n (v, w, ..., z),

p bildet also die erste Benennung für die Liste u₁ bis u_n, die in diesem Fall die Hauptliste darstellt. Durch FORM u₁ bis FORM u_n werden benannte Unterlisten gebildet. Die für u₁ bis u_n aufgeführten Namen müssen in der Hauptliste erscheinen (s. Bild 7.4).

Wie Literale eingesetzt werden, zeigt Bild 7.5. Literale können jedoch leicht mit Listen verwechselt werden.

Durch die Parameterliste wird gleichzeitig die Reihenfolge der Parameter für den Aufruf festgelegt, in der die zugehörigen aktuellen Parameter beim Aufruf

eines Makros angegeben werden müssen. Wird die Reihenfolge nicht eingehalten, müssen die einzelnen Listenwerte benannt werden; sofern noch Unterlisten bzw. deren Werte in der vorgegebenen Reihenfolge geändert werden sollen, erfordern sie ebenfalls eine Benennung. Da die Länge einer Informationseinheit auf 160 Zeichen begrenzt ist, kann durch mehrmalige Aufspaltung mittels FORM-Befehl auch eine Parameterliste abgearbeitet werden, die insgesamt mehr als 160 Zeichen umfaßt. Der formale Parameter kann auch mit einem Wert vorbesetzt werden, indem der aktuelle Wert sofort beigegeben wird.

Dieser aktuelle Parameter wird immer dann genommen, wenn beim Aufruf des Makros dem formalen Parameter kein Wert, also kein aktueller Parameter folgt.

Die Vorbesetzung hat die allgemeine Form:

name = wert name: Name des Parameters
wert: Wert, den der formale Parameter durch den aktuellen Parameter erhalten soll

Diese Vorbesetzung ist nur in Verbindung mit dem Pseudobefehl FORM zulässig (s. Bild 7.6).

Bleibt die Reihenfolge gemäß der ursprünglichen Deklaration erhalten, kann jegliche Benennung entfallen.

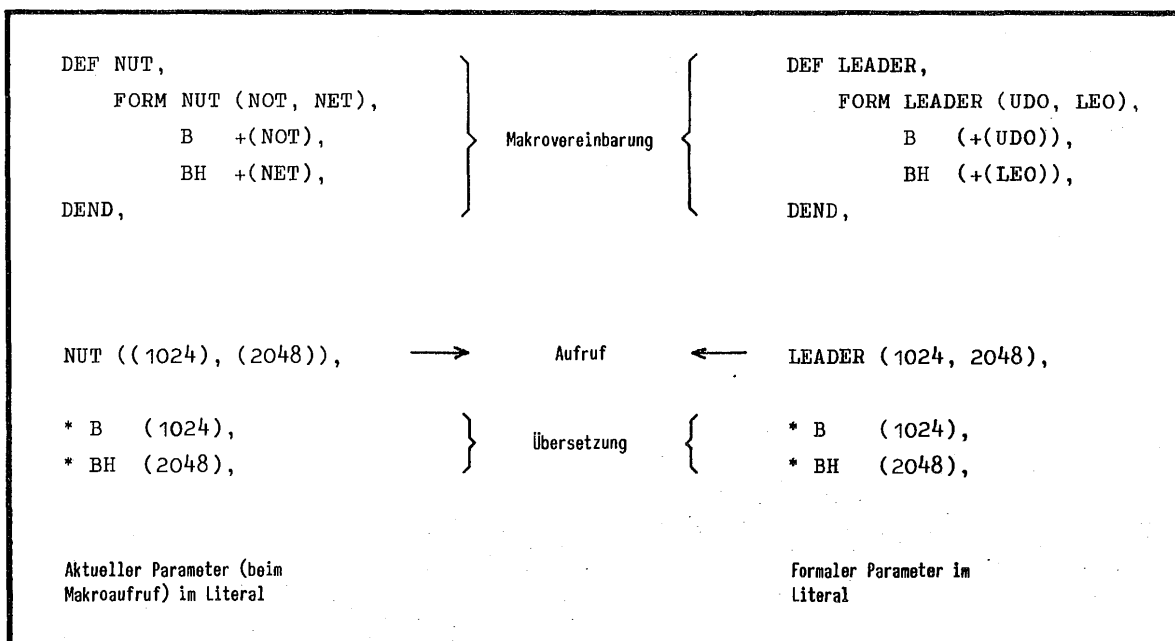


Bild 7.5 Schreibweise der Literale

```

VOR=      SEGM,
          XBasis XB,
          ASP 6K8,
          VORBES (1,'0'),
          UNTPR 5,
          ALARM FEHL,
          START ANF,
ANF.=     SEGM,
ANFANG.=  NULL 0,
          TEST*EIN,
          TEST*FR,
          NULL (ANFANG/A,
          SCHLUSS/A,'FFFFFF'/H),
          SEGM,
          DEF ANZ,
          FORM ANZ (W1,W2,X,MAX),
          FORM X (X1,X2,MIN),
          XBA +(MIN),
          XC +(X1),
          XBA +(MAX),
          XC +(X2),
          E B +(X1),
          E BH +(X2),
          C +(W1),
          CH +(W2),
          HXP 2 +(X1),
          HXP 2 +(X2),
          DEND,

```

```
ANZ (A,B,(C,D,10),100),
```

1. Aufruf

```
ANZ ANZ=(W1=A,W2=B,X=(X1=C,X2=D,MIN=10),
MAX=100),
```

2. Aufruf

```
ANZ (X=(C,D,10),100,W1=A,B),
```

3. Aufruf

```
ANZ (X=(MIN=10,X1=C,D),W1=A,B,MAX=100),
```

4. Aufruf

```

* XBA 10,
* XC C,
* XBA 100,
* XC D,
* E B C,
* E BH D,
* C A,
* CH B,
* HXP 2 C,
* HXP 2 D,

```

```
ANZ (W1=SCH,MAX=STAT,W2=ALL,X=(X1=X1,X2,LIST)),
```

5. Aufruf

```

* XBA LIST,
* XC X1,
* XBA STAT,
* XC X2,
* E B X1,
* E BH X2,
* C SCH,
* CH ALL,
* HXP 2 X1,
* HXP 2 X2,

```

```
ANZ (MAX=100,W2=Z,W1=Y,
X=(X2=L,MIN=20,X1=S)),
```

6. Aufruf

```

* XBA 20,
* XC S,
* XBA 100,
* XC L,
* E B S,
* E BH L,
* C Y,
* CH Z,
* HXP 2 S,
* HXP 2 L,

```

TAS - Handbuch

Bild 7.4 Beispiel für ein Programm mit Formatdeklaration in Form von mehreren Listen und verschiedenen Variationen des Aufrufes

1. Aufruf: Reihenfolge (für den Aufruf und seiner Zuordnung) eingehalten, Benennung nicht erforderlich, wenn sämtliche formalen Parameter beinhaltet sind.
2. Aufruf: Nur zum Vergleich zum ersten Aufruf wurde hier eine Benennung angegeben. Die Wirkung ist die gleiche.
3. Aufruf: Reihenfolge nicht eingehalten, Benennung daher erforderlich.
4. Aufruf: Reihenfolge nicht eingehalten, Benennung erforderlich. Reihenfolge der Unterlisten ebenfalls nicht eingehalten, Benennung ist also auch hier anzugeben.

Die ersten vier Aufrufe ergeben in der Übersetzung den gleichen Text (mit * gekennzeichnet).

5. Aufruf: Die Reihenfolge der Haupt- und der Unterlisten entsprechen nicht der Reihenfolge wie vorgegeben.

6. Aufruf: Ähnlich dem 5. Aufruf gehalten.

Der 5. und 6. Aufruf ergeben die im Bild gesondert aufgeführten Texte aus den Übersetzungen (mit * gekennzeichnet).

Weitere Möglichkeiten ergeben sich durch die Anwendung der Pseudobefehle VERS, SONST und VEND (s. Abschnitt 7.6. ff.).

```
DEF MAKRO2,
    FORM MAKRO2 (PARA=20),
    B (+(PARA)),
END,
```

Makrodefinition mit Makrotext und Pseudobefehl FORM (Formatdeklaration) und vorbesetztem Parameter.

```
MAKRO2,
MAKRO2 10,
MAKRO2 (10),
MAKRO2 (PARA= 'KONST'),
MAKRO2 (PARA= 'FFFFFF'),
```

- 1. einfacher Aufruf
 - 2. abgewandelter einfacher Aufruf
 - 3. Aufruf mit Änderung der Vorbesetzung
- } ähnlich dem 3. Aufruf

```
B ( 20),
B ( 20),
B ( 10),
B ('KONST'),
B ('FFFFFF'),
```

} ergibt Übersetzung nach der Voreinstellung

} Übersetzung entsprechend den Aufrufen

Bild 7.6 Makro mit Pseudobefehl FORM (Niederschrift und Übersetzung)

7.5. Wiederholungen (WIED-WEND)

Mit Hilfe des Pseudobefehls WIED lassen sich sowohl innerhalb als auch außerhalb von Makrotexen bestimmte Folgen von Informationseinheiten wiederholen. Die Anzahl der Wiederholungen wird durch einen Parameter (formalen Parameter) und seinen Zuordnungen (aktuelle Parameter) geregelt. WIED verlangt zwingend einen (und nur einer ist zugelassen) Benennungsteil:

para = WIED (p₁, p₂, ..., p_n), para: Name des formalen Parameters (Makrovariable)

text, p: aktueller Parameter für die Wiederholungen (Makrokonstante)

WEND, text: Makrotext, der wiederholt werden soll

Die Teile innerhalb der Einleitung (durch WIED) bis zur Beendigung (durch WEND) müssen den Anforderungen der bereits beschriebenen Makrotexen genügen.

Die Pseudobefehle besagen, daß der Makrotext vom Pseudobefehl WIED bis WEND wiederholt werden soll, wobei dieser Text auch als Wiederholungstext bezeichnet wird.

7.5.1. WIED-Befehl

Der WIED-Befehl leitet den Wiederholungstext ein. Der Adressenteil ist eine Werteliste, die einzelne aktuelle Parameter für die formalen Parameter enthält. Der Befehl ist eine Anweisung an den Assembler, den Wiederholungstext so oft nacheinander zu interpretieren, wie der Werteliste aktuelle Parameter enthält. Er ordnet außerdem dem formalen Parameter den entsprechenden aktuellen Wert der Werteliste zu, der ggf. durch den Pseudobefehl FORM noch aufgliedert werden kann.

Hinter dem Pseudobefehl WIED werden die aktuellen Parameter (Makrokonstante) für die einzelnen Wiederholungen angegeben. Sie sind in Klammern zu setzen und durch Komma zu trennen. Innerhalb dieses Klammerpaares sind weitere Klammerpaare möglich. Die Zahl der Wiederholungen richtet sich nach der Anzahl der Parameter. Parameter sind wie Parameter in sonstigen Makros zu behandeln.

Wenn der Wiederholungstext unverändert bleiben soll, kann trotzdem ein formaler Parameter eingeführt werden. Es erfolgt lediglich kein Bezug auf den formalen Parameter des Wiederholungstextes.

Außerdem ist es möglich, eine Makrovariable außerhalb von einer Makrodefinition einzuführen, auf die z. B. mittels der Spezifikation VERSION im UEBERSETZE - Kommando (s. Absatz 7.7.1.) und dem entsprechenden Befehl FORM VERSION * Bezug genommen wird.

Durch einen WIED-Befehl definierte Makrovariable sind bekannt:

- 1) nur innerhalb eines Makroaufrufs wenn der WIED-Befehl innerhalb einer Makrodefinition stand
- 2) im ganzen Programm wenn der WIED-Befehl außerhalb einer Makrodefinition stand.

7.5.2. Wiederholungstext

Der Wiederholungstext ist eine Folge von Informationseinheiten und darf selbst wieder Wiederholungstext-, Versionstext- und Makrodefinitionen enthalten. Diese Definitionen müssen jedoch alle im Wiederholungstext enthalten sein. Ein Wiederholungstext darf daher z. B. keinen einzelnen VERS-Befehl ohne den entsprechenden VEND-Befehl enthalten, da zu einer vollständigen Versionstextdefinition ein VERS- und ein VEND-Befehl gehören. Selbstverständlich darf im Wiederholungstext auf formale Parameter mittels der dafür geeigneten Befehle (z. B. FORM-Befehl, VERS-Befehl) Bezug genommen werden. Bei der Interpretation des Wiederholungstextes werden die im Wiederholungstext enthaltenen Variablen durch ihre aktuellen Werte ersetzt.

Innerhalb einer Makrodefinition mittels WIED-Befehl eingeführte Makrovariable können auch außerhalb des entsprechenden Makroaufrufes zugänglich gemacht werden, wenn der Name der Makrovariablen mit einem Punkt abgeschlossen wird. (Bild 7.7).

```
DEF MAK,  
    FORM MAK (A, B, C),  
    :  
    :  
ABC.= WIED (L),  
    :  
    :  
    WEND,  
DEND,
```

Bild 7.7 Innerhalb einer Makrodefinition mittels WIED - Befehl eingeführte Makrovariable

7.5.3. WEND-Befehl

Dieser Befehl schließt den Wiederholungstext ab.

Mögliche Anwendungen ergeben sich aus den Beispielen in den Bildern 7.8, 7.9 und 7.10.

Niederschrift

```

DEF TRANS,
  A= WIED +(TRANS),
  C +(A),
  WEND,
DEND,

```

Aufruf

```

TRANS (SP1, SP2, SP3),

```

Übersetzung

```

* C SP1,
* C SP2,
* C SP3,

```

Bild 7.8 Wiederholungstextdefinition innerhalb eines Makros

Niederschrift

```

A= WIED (1, 2),
B= WIED (3, 4),
  B +(A)+(B),
  WEND,
  WEND,

```

Übersetzung

```

* B 13,
* B 14,
* B 23,
* B 24,

```

Bild 7.9 Wiederholungstextdefinition außerhalb eines Makros ohne Formatdeklaration

Niederschrift

```

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

```

Übersetzung

```

* B 3,
* C 5,
* B 7,
* C 8,

```

Bild 7.10 Wiederholungstextdefinition außerhalb eines Makros mit Formatdeklaration

7.6. Versionen

Wie unter Abschnitt 7.1 bereits aufgeführt, kann der Benutzer an jeder Stelle im Quellenprogramm Bedingungen (Versionen) stellen, durch die nur bestimmte Programmteile oder Folgen von Informationseinheiten verarbeitet werden. Der Benutzer erreicht dies mit Hilfe von Versionstextdefinitionen unter Anwendung der Pseudobefehle VERS, SONST und VEND. Auch hier sind ggf. den formalen Parametern aktuelle Werte zuzuweisen.

Allgemeine Form:

```

VERS (f),
  text,
VEND,

```

f: Name des formalen Parameters oder der Bedingungsliste
text: Text (Versionstext) mit Berücksichtigung der zu stellenden Bedingung(en)

Dem Namen des formalen Parameters (f) kann auch sofort ein aktueller Wert beigegeben werden. Der Versionstext darf weitere Versionstext- oder Wiederholungs- und Makrotextdefinitionen enthalten. Sie müssen jedoch alle im Versionstext erscheinen. Ist keine der Bedingungen erfüllt, so wird der Versionstext überlesen. Der Assembler untersucht in diesem Fall nur die folgenden VERS- und VEND-Befehle, um das Ende des Versionstextes feststellen zu können. Wenn der Versionstext interpretiert wird, werden alle darin enthaltenen Pseudobefehle normal ausgeführt und im Versionstext enthaltene formale Parameter durch ihre aktuellen Werte ersetzt. Insbesondere werden in einem Versionstext enthaltene weitere Versionstextdefinitionen normal interpretiert (damit lassen sich UND-Verknüpfungen von Bedingungen durch Verschachtelungen von Versionstextdefinitionen und ODER-Verknüpfungen durch Zusammenfassung von Bedingungslisten realisieren).

Eine Bedingung 2. Art besteht nur aus einem variablen Namen. Sie ist bereits erfüllt, wenn dem Namen überhaupt ein beliebiger Wert zugeordnet ist.

Allgemeine Formen:

GL:	gleich
UG:	ungleich
LKG:	linksbündig kleiner/gleich
LKL:	linksbündig kleiner
LGG:	linksbündig größer/gleich
LGR:	linksbündig größer
RKG:	rechtsbündig kleiner/gleich
RKL:	rechtsbündig kleiner
RGG:	rechtsbündig größer/gleich
RGR:	rechtsbündig größer

7.6.1. VERS-Befehl

Der VERS-Befehl leitet den Versionstext ein. Er ist eine Anweisung an den Assembler, den folgenden Text nur zu übersetzen, wenn mindestens eine von mehreren im VERS-Befehl angegebenen Bedingungen erfüllt ist. Diese Bedingungen sind in einer Bedingungsliste zusammengefaßt. Diese Liste bildet den Adressenteil des VERS-Befehls. Es gibt 2 Arten von Bedingungen:

Eine Bedingung 1. Art wird durch eine Makrovariable angegeben, der ein Wert ggf. mit Bedingungsoperator zugeordnet ist. Sie ist erfüllt (falls kein Bedingungsoperator vorliegt), wenn der aktuelle Wert der Makrovariablen mit dem des vorbesetzten übereinstimmt.

Die Bedingungsoperatoren vergleichen die Werte mit den in der Bedingungsliste des die Version steuernden Befehls angegebenen Werten. Diese Prüfung kann man auf Gleichheit oder Ungleichheit oder auf die Bedingung "kleiner (gleich)" oder "größer (gleich)" von entweder linksbündig oder rechtsbündig gedachten Zeichenfolgen vornehmen lassen.

7.6.2. SONST-Befehl

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

7.6.3. VEND-Befehl

Der VEND-Befehl schließt einen Versionstext ab.

7.6.4. Beispiele und Erläuterungen

Niederschrift

```

DEF BERTA,
  FORM BERTA (VON,BIS,VZ=UNDEF),
  VERS (VZ=UNDEF),
    B +(VON),
  VEND,
  VERS (VZ=N),
    BN +(VON),
  C
  + (BIS),
DEND,

```

----- Makrodefinition unter dem Namen BERTA
 ----- Parameterliste BERTA (Wert VZ erhält als Vorbesetzung den Wert UNDEF)
 ----- Übersetzung, wenn Bedingung VZ = UNDEF erfüllt
 ----- Übersetzung, wenn Bedingung VZ = N erfüllt
 ----- außerhalb des Versionstextes stehende Informationseinheit
 ----- Ende der Makrodefinition

```

BERTA ((20),SP4,N),
BERTA ((20),SP4),
BERTA (VZ=M,BIS=SP4,VON=(20)),

```

1. Aufruf
2. Aufruf
3. Aufruf

Übersetzung:

```

* BN (20),
* C SP4,
* B (20),
* C SP4,
* C SP4,

```

Bild 7.11 Anwendung der Pseudobefehle VERS und VEND

```

VERS (A=3, B=4),
  VERS (C=5, D=7),
    B (30),
  VEND,
VEND,

```

Der Text B (30) wird nur übersetzt, wenn die Bedingung $((A=3) \vee (B=4)) \wedge ((C=5) \vee (D=7))$ erfüllt ist.

Bild 7.12 Anwendung einer verschachtelten Version

Niederschrift:

```

DEF MAKRO,
  FORM MAKRO (A,B),
  VERS (A=2),
    BA 0,
  SONST,
    BA 1,
  VEND,
  VERS (B=1),
    ZTR 3A,
  SONST,
    ZTR 2A,
  VEND,
DEND,

```

Aufrufe und Übersetzungen:

```

MAKRO (1,1),
* BA 1,
* ZTR 3A,
MAKRO (A=2,B=3),
* BA 0,
* ZTR 2A,
MAKRO (A=2),
* BA 0,
* ZTR 2A,
MAKRO (B=1),
* BA 1,
* ZTR 3A,

```

} A ≠ 2
 } B = 1
 } A = 2
 } B ≠ 1
 } A = 2
 } B ohne Zuweisung
 } A ohne Zuweisung B = 1

Bild 7.13 Anwendung der Pseudobefehle VERS, SONST und VEND

Niederschrift

```

DEF TEGEL,
  FORM TEGEL (YOU,HER,SHE,HIS),
  VERS (YOU=/LKG/4,HER=/RKG/7),
  VERS (SHE=/RGG/8,HIS=/RKL/5),
  B (512),
  SONST,
  BH (256),
  VEND,
  BQ (128),
  VEND,
DEND,

```

Aufruf und Übersetzung

```

TEGEL (''KONSTA'', '123456', 2, P),
* BH (256),
* BQ (128),

```

Aufrufe

```

TEGEL (3,6,9,4),
TEGEL (''1'',6,9,4),
TEGEL (6,6,8,7),

```

Übersetzung

```

* B (512),
* BQ (128),

```

Bild 7.14 Beispiele für die Anwendung von Bedingungsoperatoren

Das mit TEGEL definierte Makro hat verschachtelte Versionen unter Anwendung von Bedingungsoperatoren. Nicht erfüllte Bedingungen haben zur Folge, daß die unter SONST aufgeführte Befehlsfolge ausgeführt wird, desgleichen die u. U. anschließend außerhalb der Versionstextdefinition stehenden Informationseinheiten.

Im Bild 7.11 ist gezeigt, wie in einer Makrodefinition zwei Versionen alternativ verwendet werden. Die erste Version wird übersetzt, wenn dem formalen Parameter VZ der Wert UNDEF zugewiesen wurde und die zweite Version den Wert N erhält. Fehlt die Wertzuweisung beim Makroaufruf oder ist ein anderer Wert angegeben, wird nur nach SP4 abgespeichert.

Eine weitere Möglichkeit der Anwendung ergibt die verschachtelte Version (siehe Bild 7.12). Danach ist eine Version innerhalb einer Version möglich. Der Befehlsablauf gemäß Bild 7.12 erfolgt, wenn ((A=3) oder (B=4) und ((C=5) oder (D=7) gesetzt sind oder unter Verwendung logischer Zeichen dargestellt:

$$\underbrace{((A=3) \vee (B=4))}_{\text{äußere Version}} \quad \wedge \quad \underbrace{((C=5) \vee (D=7))}_{\text{innere Version}}$$

Niederschrift

```

DEF SUNNY,
  FORM SUNNY (WELL,AST),
  VERS (WELL=/GL/3),
  VERS (AST=/UG/4),
  BA 12,
  VEND,
  BA 16,
  VEND,
DEND,

```

Aufrufe

```

SUNNY (3,1),
SUNNY (WELL=3),
SUNNY (3,7),
SUNNY (AST=7,WELL=3),

```

Übersetzung

```

* BA 12,
* BA 16,

```

Aufruf und Übersetzung

```

SUNNY (3,4),
* BA 16,

```

Aufrufe

```

SUNNY (AST=7),
SUNNY (2,5),

```

Die beiden zuletzt aufgeführten Aufrufe ergeben keine Übersetzung, da keine der Bedingungen erfüllt sind. Die anderen Aufrufe sind ähnlich den links im Bild aufgeführten Aufrufen gehalten.

Der SONST - Befehl (siehe Bild 7.13) gibt ähnlich dem Beispiel in Bild 7.11 Alternativen. Die erste Version durch Aufruf MAKRO (1, 1), ist nicht erfüllt. Sie könnte nun weiter verschachtelt sein, je nach Programm kann jedoch die Angabe des Pseudobefehls SONST genügen. Im Beispiel wird daher zunächst BA 1 interpretiert. Die zweite Bedingung aber ist erfüllt (B = 1). Hier wird nun SONST übergangen.

Beispiele mit Bedingungsoperatoren zeigt Bild 7.14. Diese Versionen können nur zwischen BA 12 und/oder BA 16 entscheiden. Auch hier sind im Parameter detaillierte Werte zuzuordnen. Die Schreibweise für die einzelnen möglichen Formen sind aus Abschnitt 7.6.1. ersichtlich. Versionstexte werden wie die in den ersten Abschnitten beschrieben, behandelt. Zusätzlich erfolgt Vergleich auf gleich, ungleich, rechts- oder linksbündig usw.

D

113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200

7.7. Globale Makrovariable

Innerhalb der Makrotechnik gibt es außer den beschriebenen Makros, wobei den formalen Parametern aktuelle Werte durch den Benutzer zugewiesen werden können, globale Makrovariable, die bereits im Assembler definiert sind. Sie können aufgerufen werden, ohne daß der Benutzer irgendwelche Definitionen treffen muß.

Zur Unterscheidung zu den in den ersten Abschnitten beschriebenen Makros sind die globalen Makrovariablenamen mit einem * versehen, d. h., die für diese Variablen festgelegten Namen haben globale Bedeutung. Sie dürfen nicht für andere Makrovariable und nicht in Namen, die der Benutzer festlegt, verwendet werden.

7.7.1. VERSION*

(s. auch Komm.-Sprache TR 440 - UEBERSETZE -)

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

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

Die Erzeugung verschiedener Versionen geht folgendermaßen vor sich:

Im UEBERSETZE-Kommando kann als Wert der Spezifikation VERSION ein aktueller Parameter angegeben werden, der die gewünschte Version kennzeichnet. Dieser aktuelle Parameter wird vor der Übersetzung des Quellenprogramms der globalen Makrovariablen VERSION* als aktueller Wert zugeordnet. Die Makrovariable kann im gesamten Quellenprogramm wie ein normaler formaler Parameter verwendet werden, um die Interpretation des Quellenprogramms zu steuern.

So kann z. B. auf die Makrovariable im VERS-Befehl Bezug genommen werden oder falls als Wert der Spezifikation VERSION eine Liste angegeben wird, kann diese Liste mit Hilfe des FORM-Befehls aufgeschlüsselt werden. Die Steuerung der Interpretation eines TAS-Quellenprogramms verläuft also analog zur Steuerung der Interpretation eines Makros. In dieser Analogie entspricht das Quellenprogramm dem Makro, das UEBERSETZE-Kommando dem Makroaufruf und der Wert der Spezifikation VERSION dem Adressenteil des Makroaufrufs.

Das Bild 7. 15 zeigt wie je nach Aufruf eine unterschiedliche Befehlsfolge generiert wird.

Bei der ersten Übersetzung ist VERSION = (BS3 = NEIN) vereinbart. Im Quellprogramm wird die Bedingung BS3 = NEIN nur einmal erfüllt. Diese Version wird also angenommen. Der folgende Befehl B SP3+2 wird einassembliert. Des weiteren erscheint die Bedingung BS3 ohne zusätzliche Angabe. Der Befehl C SP1 wird daher ebenfalls assembliert. VERS mit anderen Bedingungen (z. B. BS3 = JA) werden übergangen.

Bei der dritten Übersetzung wird VERSION = (BS2) -auch hier wieder als Spezifikationswert im UEBERSETZE-Kommando gesetzt, d. h., daß diesem Wert entgegen dem ersten Aufruf keine weitere Zuordnung folgt. In diesem Fall wird, immer von der Quelle aus betrachtet, nur auf BS2 -VERS (BS2)- verglichen und der Befehl C SPO interpretiert. Die anderen Aufrufe sind ähnlich gehalten.

Daraus ergibt sich, daß im UEBERSETZE-Kommando unter VERSION angegebene Spezifikationen ohne Zuordnungen im Übersetzungsprotokoll nur Bedingungen, die ebenfalls keine Zuordnungen enthalten, interpretiert werden.

Im Gegensatz dazu werden im UEBERSETZE-Kommando unter VERSION aufgeführte Spezifikationen mit Zuordnungen im Übersetzungsprotokoll Versionsangaben mit erfüllten Bedingungen interpretiert, desgleichen Versionen ohne zusätzliche Angaben.

Für beide Fälle gilt, daß zusätzlich die Spezifikation PROTOKOLL bei UEBERSETZE mit 'O' anzugeben ist. Dadurch erscheint neben dem Übersetzungsprotokoll (auch hier entsprechende Spezifikation vorausgesetzt) die Ausgabe des Objektcodes, an Hand dessen bereits die Rückverfolgung der durchgeführten Interpretationen möglich wird. Außerdem ist eine Verfolgung durch entsprechende TEST-Befehle für das Überwacherprotokoll möglich.

7.7.2. NUMMER*

Diese globale Makrovariable kann überall in einem Quellenprogramm wie ein gewöhnlicher Parameter verwendet werden.

Als Wert wird NUMMER* die vom Assembler mitgezählte Nummer des aktuellen Makroaufrufs zugewiesen. Bei geschachtelten Makroaufrufen werden die aktuellen Nummern in einem aufgebauten Nummernkeller reserviert und entsprechend der Verschachtelung wieder entnommen.

Der Initialwert ist 0.

Spezifikationen im UEBERSETZE-Kommando:

◇UEBERSETZE, NUM.--STD-, MO=MONTOB, PR.=A'O'R, TRACE=-STD-, SPRACHE=TAS,
VERSION=(BS3=NEIN), QUELLE=/
D

◇UEBERSETZE, NUM.--STD-, MO=MONTOB, PR.=A'O'R, TRACE=-STD-, SPRACHE=TAS,
VERSION=(BS2=NEIN), QUELLE=/
D

◇UEBERSETZE, NUM.--STD-, MO=MONTOB, PR.=A'O'R, TRACE=-STD-, SPRACHE=TAS,
VERSION=(BS2), QUELLE=/
D

◇UEBERSETZE, NUM.--STD-, MO=MONTOB, PR.=A'O'R, TRACE=-STD-, SPRACHE=TAS,
VERSION=(BS2=JA), QUELLE=/
D

◇UEBERSETZE, NUM.--STD-, MO=MONTOB, PR.=A'O'R, TRACE=-STD-, SPRACHE=TAS,
VERSION=(BS3=JA), QUELLE=/
D

◇UEBERSETZE, NUM.--STD-, MO=MONTOB, PR.=A'O'R, TRACE=-STD-, SPRACHE=TAS,
VERSION=(BS3), QUELLE=/
D

Quelle:

FORM VERSION* (BS2, BS3),	VERS (BS3=JA),
VERS (BS2=JA),	B SP3,
B SP2,	VEND,
VEND,	
	VERS (BS3=NEIN),
VERS (BS2=NEIN),	B SP3+2,
B SP2+2,	VEND,
VEND,	
	VERS (BS3),
VERS (BS2),	C SP1,
C SPO,	VEND,
VEND,	

Beachte: Nur durch den Befehl DRUCK 3 wird alles protokolliert, also auch die
eingesetzten Makros und die abgelehnten Versionen.

Bild 7. 15 Beispiel für die Verwendung des globalen Makronamens VERSION*

Niederschrift

```

DEF GLOBVAR.
SIO
  BERECHNG1+(NUMMER*),
  A
  +(GLOBVAR)),
BERECHNG1+(NUMMER*)=
  BH
  (
  1945),
  SG
  BERECHNG2+(NUMMER*),
  SB
  +(GLOBVAR)),
BERECHNG2+(NUMMER*)=
  N
  0,
DEND,

```

```

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

```

```

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

```

Übersetzung

```

KUNDV.
KONSTANTE1= 1.
VARIABLE1= 1/V.

```

```

KUNDNR.
KONSTANTE2= 2.
VARIABLE2= 2/V.

```

```

GLOBVAR 13.
SIO BERECHNG13.
A (
  13).
BERECHNG13= BH (
  1945).
  SG BERECHNG23.
  SB (
  13).
BERECHNG23= N 0,

```

Bild 7.16 Beispiel für die Anwendung des globalen Makronamens NUMMER* (ohne Verschachtelung)

Niederschrift

```

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

```

```

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

```

```

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

```

Übersetzung

```

AUSSENMAKRO.
KONSTANTE1= 1.
INNENMAKRO,
KONSTANTE2= 2.
VARIABLE2= 2/V.
VARIABLE1= 1/V.

```

```

INNENMAKRO.
KONSTANTE3= 3.
VARIABLE3= 3/V.

```

```

PLUSODERMINUS 13.
SXI MINUS4,
A (
  13).
  S ENDE4,
  N 0,
  SB (
  13).
ENDE4= N 0,

```

Bild 7.17 Beispiel für die Anwendung des globalen Makronamens NUMMER *

Nach Bild 7. 16 wurden drei Makrodefinitionen vereinbart, die Namen lauten GLOBVAR, KUNDTV und KUNDNR. Nur wegen der Übersichtlichkeit sind Makrotexte und Aufrufe nicht unmittelbar aneinandergereiht. Mit der globalen Makrovariablen NUMMER * sind verschiedene Teile von Informationseinheiten in Form von formalen Parametern ersetzt. Als erstes Makro wird KUNDTV ohne Angabe eines aktuellen Wertes aufgerufen. KUNDTV war als zweites von insgesamt drei Makrodefinitionen in der Quelle aufgestellt. Bei der Interpretation des ersten Makroaufrufs wird +(NUMMER*) mit 1 besetzt und es wird KONSTANTE1 = 1 und VARIABLE1 = 1/V eingesetzt.

Als zweites Makro wird KUNDNR aufgerufen und +(NUMMER *) mit dem aktuellen Wert 2 belegt (KONSTANTE2 und VARIABLE2). Das erstdefinierte Makro ist als drittes und letztes aufgerufen, und zwar mit dem aktuellen Parameter 13. Bei Addition und Subtraktion waren als Adreßteile jeweils ein Literal angegeben, die den formalen Parameter enthielten. Aufgrund des Aufrufs wird jeweils der Wert 13 eingesetzt. Die mit NUMMER* besetzten Parameter erhalten den Wert 3 (aus BERECHNUNG1+(NUMMER*)) z. B. wird BERECHNUNG13 usw.).

Bild 7. 17 zeigt einen Makroaufruf und einen geschachtelten Makroaufruf mit entsprechender Einsetzung.

Das im Quellprogramm definierte Makro AUSSENMAKRO ruft das Makro INNENMAKRO auf. In der Einsetzung werden den dem ersten Aufruf (durch AUSSENMAKRO) folgenden Einheiten die Werte 1 zugeordnet. Durch den Zwischenaufruf INNENMAKRO erhalten die folgenden Größen den Wert 2. Nach Abschluß dieser Einsetzung wird die Interpretation von AUSSENMAKRO mit der unveränderten Wertzuordnung 1 fortgeführt.

Zusätzlich wird INNENMAKRO anschließend noch einmal, diesmal mit Wertzuordnung 3 (da dritter Aufruf) aufgeführt. Demzufolge erhält +(NUMMER*) in dem an vierter Stelle stehenden Aufruf PLUSODERMINUS den Wert 4.

7.7.3. DATUM* und GENV*

Um für eine zu übersetzende TAS-Quelle das Datum der Assemblierung und außerdem die Generations- und Versionsnummer der im UEBERSETZE-Kommando unter Quelle spezifizierten Datei als Konstante verfügbar zu machen, können die globalen Makrovariablen DATUM* und GENV* verwendet werden (Beispiel siehe Bild 7. 17 und 7. 18).

Als Werte werden diesen zugewiesen:

DATUM* : die Ausgabe des SSR 4 32 (2 bzw. 3 Ganzworte)

GENV* : das dritte Ganzwort der Dateibezeichnung (im Falle Fremdstring 6 Oktaden "0")

```

Niederschrift:
DATUM#          ''+(DATUM*)''/V,
Übersetzung:
DATUM#          ''31.03.71''/V,
ODER
Niederschrift:
BZ              (''+(DATUM*)''),
Übersetzung:
(''31.03.71'')
```

Bild 7. 18 Beispiele für die Anwendung des globalen Makronamens DATUM*

```

Niederschrift:
BH              (''+(GENV*)''),
Übersetzung:
(''000001'')
```

Bild 7. 19 Beispiel für die Anwendung des globalen Makronamens GENV*

7. 8. Beispiel mit Anwendung von Makros (s. Bild 7. 20)

Niederschrift

```

DEF    DAT,
FORM   DAT
      (DAT2, DAT4),
FORM   DAT2
      (DAT21, DAT22, DAT23, DAT24),
FORM   DAT4
      (DAT41, DAT42),

VERS  (DAT21=/GL/7, DAT22=/RKG/9),

VERS  (DAT23=ANG, DAT24=22),

BH    (
+ (DAT41)),

SONST (
BH    (
+ (DAT42)),

VEND,

VEND,

A=    WIED
      ((ON, AN), (IN, UN)),
FORM  A  (B, C),

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

DEND,
    
```

→ Vereinbarung für ein Makro durch den Namen DAT (= Makroname -und Pseudobefehl-) unter dem es aufgerufen wird

} Formatdeklaration für impliziten formalen Parameter (mit DAT)
-Makroname = Name für Hauptliste-
Weitere Eröffnungen von Parameterlisten. Nach FORM aufgeführte Namen für Unterlisten müssen mit denen in der Hauptliste enthaltenen identisch sein. Die Namen für die einzelnen Parameter der Listen sind frei wählbar.

} Erste gestellte Bedingung als äußere Version mit Bedingungsoperatoren

} Bedingung als innere Version (verschachtelte Version), ebenfalls mit Bedingungsoperatoren

} Befehl mit formalem Parameter vorbesetzt

} Bei Nichterfüllung von Bedingungen wird dieser Befehl immer ausgeführt, bzw. die anschließend deklarierten Informationseinheiten

→ Anzeige für das Ende der inneren Version

→ Anzeige für das Ende der äußeren Version

} Einleitung für Wiederholungstext (Wiederholungstextdefinition)

} Zu wiederholende Informationseinheiten gemäß der Angabe in der Parameterliste

→ Ende der Wiederholungstextdefinition

→ Ende der Definition für das Makro DAT

1. Aufruf

```

DAT
(DAT2=(7,9,ANG,22), DAT4=(1024,2048)),
    
```

2. Aufruf

```

DAT
((6,8,ERG,22), (1024,2048)),
    
```

3. Aufruf

```

DAT
((7,9,ANG,22), (1024,2048)),
    
```

Übersetzung

```

*           BH      (
*           B      ON,      1024),
*           C      AN,
*           B      IN,
*           C      UN,
    
```

4. Aufruf

```

DAT
((6,10,ERG,50), (0,2048)),
    
```

5. Aufruf

```

DAT
((6,10,ERG,50), (1024,2048)),
    
```

Übersetzung

```

*           BH      (
*           B      ON,      2,48),
*           C      AN,
*           B      IN,
*           C      UN,
    
```

Bild 7. 20 Beispiel für die Anwendung von Pseudobefehlen in Verbindung mit der Makrosprache

In den hier gezeigten Aufrufen wurde die durch die Formatdeklaration vorgegebene Reihenfolge eingehalten, so daß jegliche Listenbenennung entfallen kann. Für eine bessere Übersicht wurde sie im 1. Aufruf nochmals angegeben. Bei den ersten drei Aufrufen sind jeweils eine der äußeren und eine der inneren Versionsbedingungen erfüllt, d. h., der formale Parameter (+DAT 41)) wird interpretiert, nicht dagegen jedoch beim 4. und 5. Aufruf, so daß die unter SONST aufgeführten Informationseinheiten übersetzt werden.

8. Prozeduren für FORTRAN

Es ist oft wünschenswert, in ein Programm, das in einer problemorientierten Sprache (FORTRAN, ALGOL, COBOL usw.) geschrieben ist, Unterprogramme einzufügen, die im Assemblercode (TAS) geschrieben sind (Laufzeitoptimierung); oder umgekehrt in ein Programm, das im Assemblercode programmiert ist, z.B. aus Gründen der bequemen Ein- und Ausgabe, Unterprogramme aufzunehmen, die in einer problemorientierten Sprache geschrieben sind.

Da die Compiler feste Konventionen beim Aufbau von Unterprogrammen bzw. bei deren Aufruf beachten, müssen diese auch vom TAS-Programmierer wahrgenommen werden. Der vorliegende Abschnitt enthält nun diese Konventionen.

8.1. Aufruf einer FORTRAN-Prozedur

Eine FORTRAN-Prozedur wird ausgeführt, wenn sie über einen FUNCTION-Aufruf

name ($p_1, p_2 \dots p_n$)

name: Name der Prozedur

oder über eine CALL-Anweisung

p: aktueller Parameter

Call name ($p_1, p_2 \dots p_n$)

aufgerufen wird. (Siehe hierzu die Beschreibungen von FORTRAN)

Beim Aufruf einer FORTRAN-Prozedur innerhalb eines FORTRAN-Programmes wird vom Compiler ein Versorgungsblock aufgebaut.

2	Adresse S&CC+8, oder Fehleradresse	24	1	a	+0	b	2	Anzahl der Parameter
2	Hauptversorgung für 1. Parameter	24	Zusatzversorgung für 1. Parameter		24			
2	dto. für 2. Parameter	24	dto. für 2. Parameter		24			
2	dto. für 3. Parameter	24	dto. für 3. Parameter		24			
2	dto. für n. Parameter	24	dto. für n. Parameter		24			
t	1	24	48					

a: bei SUBROUTINE = 0
Typ der FUNCTION
(Typ s. Bild 8.10)

b: 0 = keine Aussage über Aufrufart
1 = Aufruf als SUBROUTINE
2 = Aufruf als FUNCTION

Bild 8.1 Aufbau des Versorgungsblocks

Die Adresse des Versorgungsblocks wird rechtsbündig im Register A an die aufgerufene Prozedur übergeben. Die ersten 8 Indexzellen, das sind die Zellen mit den Adressen 0 bis 7, stehen der Prozedur zur freien Verfügung. Das Register U ist mit 'FF' (255) vorbesetzt.

Der Prozeduraufruf selbst erfolgt intern mit dem Befehl

SFB name, name: Montageobjektname
der aufgerufenen Prozedur

wobei "name" mit dem Pseudobefehl EXTERN als fremder Montageobjektname gekennzeichnet werden muß.

Das erste Halbwort des Versorgungsblockes enthält die Adresse eines eigenen Fehlerprogramms oder die Adresse S&CC+8, wenn keine eigene Fehleroutine vorgesehen ist. Die Adresse S&CC+8 ist der Name eines Montageobjektes. Er muß also durch den Pseudobefehl EXTERN als solcher gekennzeichnet werden. In den ersten 4 Bits des zweiten Halbwortes ist angegeben, welcher Sprachkonvention die nachfolgende Versorgungsinformation genügt. In FORTRAN steht dort der Wert 1.

Die weiteren Ganzwörter des Versorgungsblockes enthalten Angaben zu den aktuellen Parametern. Dabei steht jeweils im linken Halbwort die Hauptversorgung. Sie informiert über den Ort des Parameters. Im rechten Halbwort (Zusatzversorgung) sind Angaben zur Art und zum Typ des Parameters gemacht. In Bild 8.2 ist aufgezeigt, welche Parameter möglich sind und wie bei den einzelnen Parametern Haupt- und Zusatzversorgung aufgebaut sind.

Das Beispiel in Bild 8.3 zeigt, wie ein solcher Prozeduraufruf als TAS-Programm geschrieben sein kann. Daneben ist das dazugehörige, in FORTRAN geschriebene Unterprogramm aufgeführt.

Beim Parameter "Marke" ist die Hauptversorgung die Programmadresse, die dieser Marke beim Übersetzen zugeordnet wurde. Bei den anderen Parametern besteht die Hauptversorgung aus einem Versorgungsbefehl (siehe dazu Abschnitt 2.5 "Unterprogrammaufruf über SFB und SFBE"), der die Adresse bzw. Anfangsadresse des Parameters ins Register B bringt. Der Versorgungsbefehl wird mit Hilfe des Befehls T zur Ausführung gebracht und die Adresse im Register B abgelegt. Alle anderen Register bleiben unverändert. Je nach Typ des Parameters handelt es sich um

Viertelwortadresse (siehe Bild 8.10)
Halbwortadresse
Ganzwortadresse

Bei mehrfacher Wortlänge ist es die Adresse des ersten Ganzwortes.

Parameter	Hauptversorgung	Zusatzversorgung		
		Schlüssel ²⁾	Typ ¹⁾	Art ⁸⁾
Unbestimmt	—	0	—	0
Ausdruck	Versorgungsbefehl: Adresse nach ³⁾ Register B	0	1 bis 8	1
Variable		0	1 bis 8	2
FUNCTION	Versorgungsbefehl: Anfangsadresse nach Register B	0	1 bis 8	3
SUBROUTINE		0	—	4
PROZEDUR		0	—	5
Marke	Programmadresse der Marke	0	—	6
Literalkonstante ⁵⁾	Versorgungsbefehl: Anfangsadresse nach Register B	0	Anzahl der Zeichen ⁴⁾	15
Feld	Versorgungsbefehl: Anfangsadresse nach Register B	1	Adresse der Feldbeschreibung ²⁾	
Feldelement	Versorgungsbefehl: Elementadresse nach Register B	2		

1) Typ siehe Bild 8.10

4) siehe interne Darstellung von
Literalkonstanten in Bild 8.4

2) Feldbeschreibung siehe Bild 8.5

3) Adresse ist vom Typ abhängig

5) Literalkonstante bei FUNCTION
nicht erlaubt

Ganzwortadresse
Halbwortadresse
Viertelwortadresse

Bild 8.2 Aktuelle Parameter im Versorgungsblock

Aufrufendes Programm

```

X=      : ASP 2/G,
        :
        : C      X,
        :
        : EXTERN AUSGAB ,
        :
VERSBLOCK.=S&CC+8/AV, '100101'/HV,
        : XBA X/V, '000302'/HV,
        :
        : EXTERN S&CC,
        :
        : BA  VERSBLOCK,
        : SFB AUSGAB ,
        :

```

Definition der
Speicherzelle X

Speichern eines
Wertes nach X

Kennzeichnen des
Namens AUSGAB
als fremden Mon-
tagenamen

Aufbau des Ver-
sorgungsblockes

Kennzeichnen des
Namens S&CC als
Name eines Mon-
tageobjektes

Anspruch des
Unterprogramms

```

SUBROUTINE AUSGAB (A)
WRITE (6,10) A
FORMAT (E20.12)
RETURN
END
10

```

Bild 8.3 Prozeduraufruf als TAS-Programm

Die Zusatzversorgung gibt an, welcher Art der Parameter ist, und bei einigen Parametern wird außerdem der Typ des Parameters angegeben. Der Typ gibt dabei an, welche Wortlänge der Parameter hat und damit gleichzeitig welche interne Darstellung. In Bild 8.10 sind die möglichen Typen aufgeführt und die interne Darstellung gegeben.

Beim Parameter "Literalkonstante" steht in der Zusatzversorgung anstelle des Typs wieviele Zeichen die Literalkonstante enthält. Die gesamte Länge der Literalkonstanten beträgt immer das Vielfache von 4 Ganzwörtern. Die Typenkennungen der Ganzwörter sind abwechselnd 0 und 1; das erste Ganzwort in der Literalkonstanten hat die Typenkennung 0, das letzte immer 1. Die interne Darstellung der Literalkonstanten ist in Bild 8.4 gezeigt. Die Zeichen sind in 8 Bits dargestellt, gemäß dem Zentralcode, und werden in einem Viertelwort rechtsbündig abgelegt, so daß vier Zeichen in einem Wort stehen. Die übrigen linken vier Bits im Viertelwort sind 0. Gegebenenfalls muß der Rest der vier letzten Ganzwörter mit dem Zeichen Zwischenraum (sedezimal '0AF') aufgefüllt werden. Der Parameter "Literalkonstante" ist beim Aufruf eines FUNCTION-Unterprogramms nicht erlaubt.

Ist der Parameter ein Feld oder ein Feldelement, so steht in der Zusatzversorgung die Adresse der zugehörigen Feldbeschreibung. Der Aufbau der Feldbeschreibung ist aus Bild 8.5 ersichtlich.

										48
0	0000	1. Zeichen	0000	2. Zeichen	0000	3. Zeichen	0000	4. Zeichen		
1	0000	5. Zeichen	0000	6. Zeichen	0000	7. Zeichen	0000	8. Zeichen		
0										
1										
1										
0	0000	n-4. Zeichen	0000	n-3. Zeichen	0000	n-2. Zeichen	0000	n-1. Zeichen		
1	0000	n. Zeichen	0000	SP	0000	SP	0000	SP		

Rest mit Zwischenraum (SP) auffüllen

Zeichen gemäß Zentralcode

Länge der Literalkonstanten ist immer Vielfaches von 4 Ganzwörtern

Bild 8.4 Interne Darstellung von Literalkonstanten

Feldbeschreibung

0	8	Typ ¹⁾	8	Anzahl der Dimensionen ²⁾	8
Anfangsadresse des Feldes ²⁾					
Adresse des ersten nicht mehr benutzten Platzes ²⁾					
Adresse des Dimensionsblockes					

Dimensionsblock

Anzahl der Elemente der	1. Dimension
Anzahl der Elemente der	2. Dimension
Anzahl der Elemente der	n. Dimension

1) Typ siehe Bild 8.10

2) Adresse ist vom Typ abhängig

Die Wörter haben Typenkennung 2

Bild 8.5 Feldbeschreibung mit Dimensionsblock

Die Länge des Feldes in Viertel- oder Halbwörtern gibt die Differenz der Adressen an (3. Halbwort minus 2. Halbwort). In Bild 8.8 sind in einem Beispiel für eine zweidimensionale Matrix mit den Dimensionen (10, 10) mit der Anfangsadresse ANFAMATRX die zugehörige Feldbeschreibung und der Dimensionsblock dargestellt.

Während die Hauptversorgung angibt, wo sich der Parameter befindet, und dadurch für jeden Parameter immer erforderlich ist, wird die Zusatzversorgung nicht immer benötigt. Sie wird jedoch in jedem Fall mit angegeben. Der Programmierer der Prozedur kann also selbst entscheiden, ob er die Zusatzversorgung auswertet.

```

:
: INTEGER*2 J
:
: DIMENSION AMATRX (10,10)
:
: CALL SPUR (J,AMATRX,SP)
:
:

```

Bild 8.6 Beispiel für Aufruf einer FORTRAN-Prozedur

Aufgerufenes Programm

SPUR.=	SEGM,			Die Marke SPUR ist Unterprogrammeingang
	EINGG	SPUR,		
KB=	0/V,	0/HV,		Speicherplatz für Rücksprungadresse (2. Halbwort bleibt frei)
	TBC	KB,		Retten der Rücksprungadresse
	TRX	A	0,	Adresse des Versorgungsblocks wird nach Indexzelle 0 gebracht (zwecks Übernahme der Parameter)
	LR	0A,		Löschen des Registers A (SUM=0.)
	EZ	T	0,	Schleifenzähler (Indexzelle 1) wird auf Anfangswert gesetzt
	MAB	TCB	0,	
	VBA	1,		
	XCN	1,		
	HBA	2,		Aufbereitung der Fortschrittgröße für Schleife; die Fortschritt- größe steht in Indexzelle 2
	SHB	L	1,	
	XC	2,		
	EZ	T	0,	Voreinstellung der Schleife: Anfangsadresse des Feldes nach Register B
	MAB	GA	0,	Eigentliche Schleife
	HBPX	1	2,	
	SZX	-2	1,	
	EZ	T	0,	Rücktransport des Ergebnisses (SP=SUM)
	MAB	C	0,	
	SE	KB,		Rücksprung in aufrufende Programmeinheit

Bild 8.7 In TAS geschriebene FORTRAN-Prozedur

In Bild 8.6 ist der Aufruf der FORTRAN-Prozedur in Bild 8.7 mit den aktuellen Parametern J, AMATRX und SP gezeigt. Das Unterprogramm SPUR berechnet die Summe aus den Diagonalelementen einer zweidimensionalen Matrix. Die Parameter werden mit dem Befehl EZ T 0 jeweils geholt, wenn sie benötigt werden.

Ist das aufrufende Programm im Assemblercode (TAS) programmiert, so können in FORTRAN geschriebene Prozeduren aufgerufen werden, wenn die vorstehenden Bedingungen eingehalten werden.

Die Beispiele der Bilder 8.8 und 8.9 entsprechen denen aus Bild 8.6 und 8.7. Hier ist jedoch der Prozeduraufruf im Assemblercode und die Prozedur in FORTRAN programmiert.

Aufgerufenes Programm

```

SUBROUTINE SPUR(I,XMATRX,SP)
  INTEGER*2 I
  DIMENSION XMATRX(I,I)
  SUM = 0.
  DO 10 LAUF = 1,I
10  SUM = SUM + XMATRX(LAUF,LAUF)
  SP = SUM
  RETURN
END

```

Bild 8.9 In FORTRAN geschriebene Prozedur

AMATRX=	:			} Feldbeschreibung für A-MATRIX
	'000302'/H,			
	ANFAMATRX/A,			
	ENDAMATRX/A,			
DIMAMATRX=	10/H,			} Dimensionslock für A-MATRIX
	10/H,			
ANFAMATRX=	ASP 200/G,			} Freihalteanweisung für A-MATRIX
ENDAMATRX=	0/V,			
VB=	FEHLROUT/AV,	'100103'/HV,		} Aufbau des Ver- sorgungsblocks für Unterpro- grammaufruf
	XBA I/V,	'000201'/HV,		
	TCB AMATRX+1/V,	AMATRX/ANV,		
	XBA SP/V,	'000203'/HV,		
	EXTERN SPUR,			
	BA VB,			} Kennzeichnen des Namens SPUR als Eingangsnamen ei- nes fremden Unterprogramms
	SFB SPUR,			
	:			

Bild 8.8 Beispiel für Prozeduraufruf als TAS-Programm

8.2. Rücksprung in die aufrufende Programmeinheit

Nach Abarbeitung der Prozedur muß das übergeordnete Programm Zugriff zu den Ergebnissen haben. Bei der SUBROUTINE können diese nur über den Versorgungsblock erreicht werden, da im Versorgungsblock angegeben ist, wo sich die Parameter aufhalten, oder über Variable bzw. Felder gemeinsamer COMMON-Bereiche.

Der in einem FUNCTION-Unterprogramm errechnete Funktionswert muß, in Abhängigkeit vom Typ, im Register A, im doppelt langen Register A und Q oder im Pseudoakkumulator, beginnend bei der Adresse F&RA, hinterlassen werden. Hierüber ist in Bild 8.10 eine Aufstellung gemacht. Weitere Werte können über die Parameter bzw. über Variablen und Felder gemeinsamer COMMON-Bereiche übertragen werden.

Ist der Funktionswert vom Typ INTEGER * 2, so sollte er zweckmäßigerweise mit dem Befehl B2V ins Register A gebracht werden.

Nach Abarbeitung der FORTRAN-Prozedur müssen die Register X und U dieselben Inhalte wie beim Ansprung aufweisen. Das Register U muß nach der Rückkehr ins Hauptprogramm wieder auf 'FF' gesetzt sein.

8.3. Aufbau der Prozedur

Die Prozedur findet zu Beginn ihrer Arbeit im Register B die technische Rücksprungadresse und im Register A die Anfangsadresse des Versorgungsblocks vor.

Der erste Befehl in der Prozedur (bei mehreren Eingängen der erste Befehl jedes Einganges) muß der Befehl

```
name = TBC m      name: Name der Prozedur
                  m: Anfangsadresse des Kontrollblocks
```

sein. Er legt die technische Rücksprungadresse in einem Halbwort ab. Falls in der Prozedur eine Fehlerbehandlung vorgesehen wird, werden außer diesem Halbwort

	Typ	interne Darstellung	Register
0	unbestimmt	—	—
1	integer * 2	Festkommazahl halber Wortlänge	$\langle A \rangle_{25-48}$
2	integer * 4	Festkommazahl einfacher Wortlänge	$\langle A \rangle$
3	real * 4	Gleitkommazahl einfacher Wortlänge	$\langle A \rangle$
4	real * 8	Gleitkommazahl doppelter Wortlänge	$\langle A, Q \rangle$
5	complex * 8	zwei Gleitkommazahlen einfacher Wortlänge	Pseudoakkumulator ¹⁾
6	complex * 16	zwei Gleitkommazahlen doppelter Wortlänge	Pseudoakkumulator ²⁾
7	logical * 1	Viertelwort (TK=1) ^{3) 4)}	$\langle A \rangle_{25-48}$
8	logical * 4	Festkommazahl einfacher Wortlänge ³⁾	$\langle A \rangle$

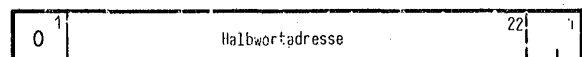
1) F&RA, F&RA+2

2) F&RA, F&RA+2, F&RA+4, F&RA+6

3) nur rechtes Bit von Bedeutung

FALSE = 0
TRUE = 1

4) Viertelwortadresse:



0: linkes Viertelwort
L: rechtes Viertelwort

Bild 8.10 Ergebnisort für die FUNCTION

drei weitere Halbwörter gebraucht. Dieses eine bzw. die vier Halbwörter werden als Kontrollblock bezeichnet. Das erste Wort des Kontrollblocks nimmt also die technische Rücksprungadresse auf.

In Bild 8.12 sind die zwei möglichen Formen des Kontrollblocks gezeigt. Die Fehlerbehandlung und der Aufbau der dazugehörigen drei Halbwörter des Kontrollblocks sind im Abschnitt 8.6 behandelt.

Die Anfangsadresse des Versorgungsblocks steht beim Aufruf der Prozedur im Register A. Ihre Sicherstellung und die Übernahme der Parameter kann beliebig erfolgen. Es ist dabei zu beachten, daß die Adressen 22 Bits lang sein können.

Ist der Variablenname in Schrägstriche eingeschlossen, oder ist der Formalparameter ein Feld, so wird der Wert des Aktualparameters zu Anfang der Prozedur nicht übernommen; es wird dann immer auf die Adresse des Aktualparameters abgespeichert. Die Werte von Aktual- und Formalparametern sind somit während der Ausführungszeit des Unterprogrammes gleich.

Für Konstanten und Zwischenergebnisse usw., die nur von der Prozedur benötigt werden, muß die Prozedur selbst Festlegungen treffen. Dies gilt auch für den Kontrollblock.

Des weiteren belegt die Prozedur mit ihren Befehlen auch den B-Bereich.

In Bild 8.11 ist in einem Beispiel das oben Beschriebene dargestellt.

8.4. Speicherbedarf

Für den Speicherbedarf der Prozedur gilt das in Abschnitt 1.4 "Prinzip der Unterprogramme" Gesagte.

Speicherbereiche, die die Daten enthalten, die von der Prozedur verarbeitet werden sollen, sind bereits im übergeordneten Programm festgelegt und brauchen daher von der Prozedur nicht berücksichtigt zu werden. Das gleiche gilt für Speicherbereiche, in denen die Ergebnisse abgelegt werden, die vom übergeordneten Programm erwartet werden und die durch das Unterprogramm erarbeitet wurden.

Es gibt für die Verarbeitung der Variablen in der Prozedur zwei Möglichkeiten. Die eine Möglichkeit besteht darin, daß die Prozedur zunächst nur mit dem Wert des Formalparameters arbeitet. Erst vor dem Rücksprung ins Hauptprogramm wird die Adresse des Aktualparameters geholt und das Ergebnis dem Aktualparameter zugeordnet, d.h., der Wert des Aktualparameters bleibt bis vor die Rückkehr erhalten. Erst dann wird er durch den Wert des Formalparameters (das inzwischen erreichte Ergebnis) ersetzt. Aktual- und Formalparameter sind während der Prozedur verschieden. Wird die Prozedur in FORTRAN geschrieben, so gilt das soeben Beschriebene für Variablen, deren Variablenname nicht in Schrägstriche eingeschlossen ist.

8.5. Indexzellen

Für Prozeduren sind vom Hauptprogramm die Indexzellen 0 bis 7 freigehalten. Diese Indexzellen stehen der Prozedur ohne Einschränkung zur Verfügung. Sie können jedoch beim Aufruf einer Prozedur beliebige Werte enthalten, da ja alle Prozeduren diese Indexzellen benutzen können. Andere Indexzellen können nicht benutzt werden, da unbekannt ist, ob diese vom Hauptprogramm benutzt werden und wichtige Daten enthalten können.

Eine weitere Möglichkeit besteht darin, in der Prozedur eine neue Indexbasis festzulegen. Damit stehen der Prozedur 256 Indexzellen zur Verfügung, die sie beliebig verwenden kann. Vor der Rückkehr in das übergeordnete Programm muß jedoch die alte Indexbasis wieder hergestellt werden. Dies gilt auch, wenn ein Fehler aufgetreten ist und die Fehleradresse angesprungen wird. Die Prozedur muß für den eigenen Indexbereich selbst einen Speicherplatz festlegen (durch Pseudobefehl ASP). Dieser Speicherraum bleibt ihr erhalten, so daß die Prozedur bei einem späteren Aufruf alle Indexzellen unverändert wieder vorfindet. Einzelheiten zur Indexbasisumschaltung sind dem Abschnitt "Befehle und Adressierung" zu entnehmen.

Aufrufendes Programm

```

:
:
: DIMENSION CFELD(10)
:
: CALL UP(A,B,CFELD,&4000,4HENDE)
:
: ...
: CALL UP1
:

```

Die zu verwirklichende SUBROUTINE-
anweisung soll folgendes Aussehen
haben:

SUBROUTINE UP(X, |Y|, ZFELD, *, U)

UP.=	SFGM,	
	EINGG UP,	
	TBC KB,	Retten der Rücksprungadresse für Normalrückkehr
	TRX A 0,	für Übernahme der Parameter
	TBC VERSBLOCK,	Retten der Anfangsadresse des Versorgungsblocks
KB.=	1/2V,	Kontrollblock für Fehlerfall
	'810A0A'/HLV, ADRFTEXT/AV,	
X=	EZ T 0,	Typabhängige Übernahme des 1. Parameters; benennen der Indexzelle 1 und abspeichern der Adresse des Parameters in diese Indexzelle
ADRX=	MAB B 0, O/V,	
	GLCH 1,	Benennen der Indexzelle 2 und abspeichern der Adresse des 2. Parameters in diese Index- zelle; kein Transport, da "Nameparameter"
	XC ADRX, C X,	
ADRY=	EZ T 0,	Benennung der Indexzelle 3 und abspeichern der Adresse des 3. Parameters in diese Index- zelle; kein Transport, da felder prinzipiell "Name parameter" sind
	GLCH 2, XC ADRY,	
ADRZFELD=	EZ T 0,	Retten der Rücksprungadresse für besonderen Rücksprung (Übergebenes Label)
	GLCH 3, XC ADRZFELD,	
RUECK4000=	EZ TCB 0, O/HV,	Übernahme des 5. Parameters; benennen der Indexzelle 4 und abspeichern der Adresse dieses Parameters in die Indexzelle 4
	TBC RUECK4000,	
U=	EZ T 0,	Rückspeichern der "Valuepara- meter" und Normalrückkehr
ADRU=	O/V, MAB B 0,	
	GLCH 4,	Rückspeichern der "Valuepara- meter" und Rücksprung auf Übergebenes Label
	XC ADRU, C U,	
	...	
	B X,	
	MF ADRX,	
	C 0,	
	B U,	
	MF ADRU,	
	C 0,	
	SE KB,	
	...	
	B X,	
	MF ADRX,	
	C 0,	
	B U,	
	MF ADRU,	
	C 0,	
	SE RUECK4000,	
	...	
	VERSBLOCK=O/HV,	
	ADRFTEXT= 'EIGENER FEHLERTEXT *037'',	
	BH (O/HG, UP/A),	Anspruch des Fehlerausgangs
	SE VERSBLOCK,	

Bild 8.11 Beispiel für Übernahme und Rückspeicherung der Parameter

8.6. Fehlerbehandlung

Ist in der Prozedur eine Fehlerbehandlung vorgesehen, so werden für den Kontrollblock 4 Halbwörter benötigt. Er hat den im Bild 8.12 gezeigten Aufbau.

Als Fehlerschlüssel ist der Wert 1 anzugeben und als Prozedurschlüssel der Wert 2570 (sedezimal 'A0A'). Im 4. Halbwort ist die Adresse eines Fehlertextes anzugeben.

Aufbau des Kontrollblocks:

2	RAS		24
	FS		24
2	1	SS	7
	US		16
	FT		24

Bild 8.12 Kontrollblock bei Fehlerbehandlung

- RAS Rückkehradresse im Normalfall
- FS Fehlerschlüssel, in Codeprozeduren ist FS = 1 zu verwenden
- SS Sprachschlüssel (SS = 1 für FORTRAN)
- US Unterprogrammschlüssel, US = 'AOA' für Benutzerprozeduren
- FT Adresse des benutzereigenen Fehlertextes

Bei Sprung auf den Fehlerausgang ist im H-Register rechtsbündig die Adresse des Befehls, den die Rückkehradresse in den Kontrollblock speichert, mitzuliefern.

Der Ansprung der Fehleradresse bewirkt, daß im Protokoll ein Fehlertext ausgegeben wird. Danach wird angegeben, an welcher Stelle im FORTRAN-Programm die Prozedur aufgerufen wurde. Der Programmlauf wird in jedem Fall abgebrochen.

Der Fehlertext muß im Zentralcode vorliegen und mit der Oktade TE (sedezimal '25') abgeschlossen sein.

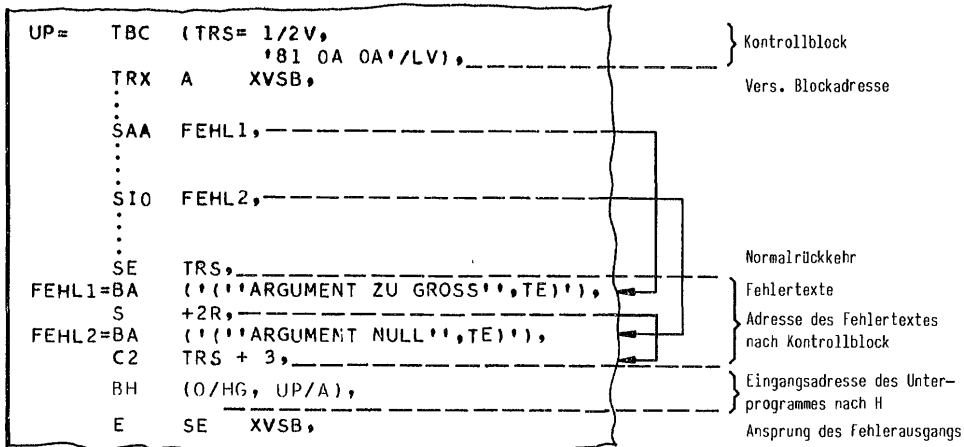


Bild 8.13 Beispiel für Fehlerbehandlung

9. PROZEDUREN FÜR ALGOL

Für den Anschluß von in TAS geschriebenen code-Prozeduren gelten die in diesem Kapitel beschriebenen Bedingungen. Im ALGOL-Programm steht lediglich der Prozedurkopf, anstelle des Prozedurrumpfs steht das Wortsymbol code.

Vom ALGOL-Übersetzer werden dem Montierer für jede code-Prozedur zwei Informationen mitgeteilt:

1. Der Name des Montageobjekts.
2. Der Name des Eingangs in die Prozedur; dieser Externbezug ist in der code-Prozedur durch den entsprechenden EINGG-Pseudobefehl abzusättigen.

Der Name des Montageobjekts und des Eingangs in die code-Prozedur sind gleich dem Namen dieser Prozedur.

TAS-Unterprogramme, die nach FORTRAN-Konventionen codiert sind, können durch das Wortsymbol fortran statt code angeschlossen werden.

Der Aufruf einer code-Prozedur unterscheidet sich nicht vom Aufruf einer vorübersetzten ALGOL-Prozedur, in den Registern werden folgende Informationen übergeben:

- $\langle B \rangle$ = Rückkehradresse aus der Prozedur im Normalfall
- $\langle A \rangle_{25-48}$ = Adresse des Versorgungsblocks
- $\langle Q \rangle_{25-48}$ = Basisadresse der Aufrufhierarchie, d. i. die zum Zeitpunkt des Aufrufs eingestellte Indexbasis
- $\langle H \rangle_{25-48}$ = Basisadresse für den Globalvektor; für code-Prozeduren ohne Bedeutung

In ALGOL-Objektläufen wird beim Prozeduraufruf der für Prozeduren benötigte Arbeitsspeicher eingerichtet und die Indexbasis auf dessen Anfangsadresse gesetzt (siehe Abschnitt 9.2 Dienste der Freispeicherverwaltung).

Der aktuelle Freispeicherpegel steht im linken Halbwort von S&C5 (1. GW der C-Zone S&CA), im rechten Halbwort von S&C5 steht der Anfangswert des Unterprogrammordnungs Zählers ($U_0 = 17$).

9.1. Verwendung der Indexspeicher

Jede Aktivierung einer ALGOL-Prozedur besitzt eine eigene Indexbasis. Die Verwendung der Indizes 0 - 17 ist festgelegt.

0 - 7 werden freigehalten (Programmiersystemkonvention)

- 8 Rückkehradresse im Normalfall
- 11 Adresse des Versorgungsblocks
- 12 Basisadresse für die Berechnung eines aktuellen Parameters; wird vor dem Aufruf eines Parameters geladen
- 13 Basisadresse der Aufrufhierarchie; die vor dem Aufruf eingestellte Indexbasis
- 14 letzte Indexbasis } werden beim
15 letzter Unterpr. Ord.-zähler } BCI getauscht
- 16 aktuelle Basisadresse
- 18 ff Rückkehradresse aus Unterprogrammen, die mit SU oder SUE aufgerufen werden

Der Unterprogrammordnungs Zähler wird beim Umstellen der Indexbasis auf $U_0 = 17$ initialisiert.

Diese Verwendung der Indizes wird auch bei der Rückverfolgung im Fehlerfall vorausgesetzt.

Die Indexzellen von 0-7 werden von Versorgungsunterprogrammen nicht zerstört.

Die auf 17 folgenden Indexzellen dürfen für lokale Zwecke frei verwendet werden.

9.2. Dienste der Freispeicherverwaltung

Das Anfordern und das Freigeben des für eine Prozedur benötigten Freispeichers geschieht durch den Aufruf der Prozedur A&FSP (Freispeicherverwaltung).

- Anfordern von Freispeicher:

$\langle A \rangle = 1$

$\langle H \rangle$ = Länge des angeforderten Speichers in Ganzworten

Bei einer Verlängerung des Freispeichers wird eine Reserve von einer Achtelseite mit eingeplant.

- Speicherfreigabe

$\langle A \rangle = 3$

$\langle H \rangle$ = Adresse, auf die der Freispeicherpegel zurückgesetzt werden soll

Bei jedem Aufruf einer ALGOL-Prozedur wird der für die Prozedur benötigte Speicher angefordert und die Indexbasis auf dessen Anfangsadresse gesetzt.

Eventuell notwendige Gebietsmanipulationen (Verlängern des Freispeichers) werden von der Freispeicherverwaltung abgehandelt.

9.3. Aufbau des Versorgungsblocks

Der Versorgungsblock eines Prozeduraufrufs mit k Parametern umfaßt k + 1 Ganzworte, jedem Parameter ist ein Ganzwort zugeordnet.

	2 ₄	4	8
TK2	FA	SS	PZ
TK2	HV1	ZV1	
TK2	HV _k	ZV _k	

- FA Fehlerausgang
 SS Sprachschlüssel (SS = 2 für ALGOL 60)
 PZ Parameterzahl
 HV_i Hauptversorgung
 ist ein Befehl, der die Berechnung des aktuellen Parameters bewirkt (siehe Abschnitt 9.10 Leistung der Versorgungsunterprogramme).
 ZV_i Zusatzversorgung
 enthält weitere Information über den aktuellen Parameter (siehe Abschnitt 9.11 Aufbau der Zusatzversorgung).

9.4. Aufruf einer ALGOL-Prozedur

Beim Betreten einer ALGOL-Prozedur werden die Registerinhalte in den "Verwaltungsindizes" 8 - 17 bezüglich der für diese Prozedur neu einzustellenden Indexbasis abgelegt. Die ersten Befehle einer übersetzten ALGOL-Prozedur sind in Bild 9.1 gezeigt.

In code-Prozeduren, die das UP A&FIXP nicht verwenden, sind dessen Aufgaben direkt abzuhandeln.

9.5. Kopf einer Code-Prozedur nach ALGOL-Konvention

Außer den Indizes 0 - 17 sollen weitere 10 Ganzworte Arbeitsspeicher benötigt werden (siehe Bild 9.2).

```

BD   S&C5           --S&C5 ENTHAELT DEN FREISPEICHERPEGEL
                        IM LINKEN, U0=17 IM RECHTEN HALBWORT--
R    MCFU  DL,
QCR  8              --ABSPEICHERN DER REGISTER--
R    MCF  DL,
TBC  16            --NEUE BASISADRESSE IN DEN INDEX 16 ABLEGEN--
SFBE (A&FIXP/A)    --IN DIESEM UP ERFOLGT DAS UMSTELLEN DER
                        INDEXBASIS, EINRICHTEN DES FREISPEICHERS
                        UND UEBERTRAGEN DER PARAMETER--
-- PROZEDURRUMPF --
  
```

Bild 9.1 Übersetzte ALGOL-Prozedur

```

BD   S&C5,
R    MCFU  DL,
QCR  8,
R    MCF  DL,
TBC  16,
MAB  BCI  14  -- INDEXBASIS AUF DIE AA DES ARBEITSSPEICHERS SETZEN--
BA   19,
BAR  1,
SFBE (A&FSP/A) -- 19 GANZWORTE SPEICHER ANFORDERN --
-- PROZEDURRUMPF --
  
```

Bild 9.2 Beispiel

9.6. Zugriff auf aktuelle Parameter und lokale Größen

Im Prozedurlauf kann der i-te aktuelle Parameter durch den entsprechenden Versorgungsbefehl, der auf der Adresse $2 * i$ relativ zum Anfang des Versorgungsblocks steht, aktiviert werden.

Vorher sind der Index 12 und das Register B mit der Basisadresse der Aufrufhierarchie (Index 13) zu laden.

```
TXX 12 13,
MFU 11,
T 4 -- AKTIVIEREN DES
      VERSORUNGSBEFEHLS--
```

Bild 9.3 Beispiel für die Aktivierung des 2. Parameters

Die lokalen Größen einer code-Prozedur können im Freispeicher oder auch in einer eigenen Adreßzone (Zugriff ohne Modifizierung möglich) abgelegt werden.

9.7. Ausgang aus einer Code-Prozedur

Der angeforderte Freispeicher ist wieder freizugeben und die Indexbasis umzustellen. Ein Funktionswert ist im Register A zu übergeben (siehe Bild 9.4).

9.8. Strukturerklärung

Ein Montageobjekt, das eine ALGOL-Prozedur nach obigen Konventionen ist, muß eine Strukturklärung

STRUKT(2)

enthalten (UP nach ALGOL-Konventionen).

```
TXR H 16 -- BASISADRESSE DER CODEPROZEDUR --
BA 3,
SFBE (A&FSP/A) -- SPEICHERFREIGABE --
(B FUNKTIONSWERT),
MF 16,
BCI 14,
MAB SE 8 -- RUECKSPRUNG --
```

Bild 9.4 Ausgang aus einer Code-Prozedur

9.9. Fehlerbehandlung

Im 1. Halbwort des Versorgungsblocks wird als Fehlerausgang die Adresse S&CC+5 (Fehlereingang der Kontrollprozedur mit Kontrollblockadresse im Register H) angegeben.

Nach dem Erkennen einer bestimmten Fehlersituation ist ein Kontrollblock nach den allgemeinen Konventionen des Programmiersystems anzulegen. Dessen Adresse wird beim Sprung auf den Fehlerausgang im Register H (rechtsbündig) übergeben.

Aufbau des Kontrollblocks

TK2	RAS		
TK2	FS		
TK2	¹ F	⁷ SS	¹⁶ US
TK2	FT		

RAS Rückkehradresse im Normalfall

FS Fehlerschlüssel, in code-Prozeduren ist FS = 1 zu verwenden

F Fortsetzungsgebiet, F = L wenn das 4. Halbwort auszuwerten ist

SS Sprachschlüssel (SS = 2 für ALGOL 60)

US Unterprogrammsschlüssel, US = 'AOA' für Benutzerprozeduren

FT Adresse eines für den speziellen Fehlerfall auszugebenden Textes

Die Informationen im Kontrollblock dienen

- zum Druck des entsprechenden Fehlertestes
- zum Start der Rückverfolgung

In dem Beispiel einer code-Prozedur im Bild 9.5 soll geprüft werden, ob der Aufruf mit 3 Parametern erfolgt ist.


```

KB= 1/V2,
    '820A0A'/V2,
    FTEX/AV,
-- KONTROLLBLOCK, BIS AUF RAS FEST INITIALISIERT --

FTEX= ''PARAMETERZAHL DER FUNKTION'',
      ''FKT UNGLEICH 3 * 037'',
      :
-- PRUEFUNG AUF PARAMETERZAHL 3 --
MF 11,
B3 1          -- PZ AUS DEM VB --
BAR 3,
SN FEPZ      -- FEHLER: PZ NICHT = 3 --
:
-- FEHLERBEHANDLUNG --
FEPZ= XB 8,
      TBC KB          -- RUECKKEHRADRESSE IN KB EINTRAGEN --
      BH (O/HG, KB/A) -- KB-ADRESSE IN H UEBERGEHEN --
      MF 16,
      BCI 14         -- LETZTE INDEXBASIS EINSETZEN --
      MAB MCF 11,
      SE 0           -- FEHLERAUSGANG --

```

Bild 9.5 Beispiel für Fehlerbehandlung

Beim Auftreten des Fehlerfalls wird im Ablaufprotokoll folgender Text eingetragen:

BENUTZERPROGRAMM EIGENER FEHLERTEXT:
PARAMETERZAHL DER FUNKTION FKT UNGLEICH 3

Bemerkung:

Der eigene Fehlertext darf max. 78 Zeichen lang sein und ist mit einer TE-Oktade (#037) abzuschließen.

Der Sprung auf eine formale Marke ist also durch folgende Befehle zu realisieren:

```

TXX 12 13,
MFU 11,
T 2*i,
CH HILFSSPEICHER,
ZI HILFSSPEICHER+1,
MABI S 0,

```

- AP ist ein switch-Name
Sprung auf die switch-Komponente, deren Nummern (als Gleitkommazahl) in A steht
- AP ist der Name einer parameterlosen Funktionsprozedur
<A> = Funktionswert

Der Funktionsaufruf geschieht im Versorgungsunterprogramm.

- AP ist der Name einer eigentlichen Prozedur oder einer Funktionsprozedur mit Parametern
 = Startadresse der Prozedur
<H> = Basisadresse der Hierarchie, aus der der Globalvektor zu übertragen ist

Für den formalen Prozeduraufruf aus einer code-Prozedur ist ein Versorgungsblock anzulegen, der Aufruf erfolgt dann durch

```

TXR Q 16,
TXX 12 13,
BA VERSBL --ADRESSE DES VBL--
MFU 11,
T 2*i,
MABI SFB 0 --SPRUNG AUF DIE
            STARTADRESSE,
            EVT. IN ANDERE
            GROSSEITE--

```

- AP ist ein string
 = Adresse des Kopfwortes
<A> = Kopfwort des string, d. i. die Länge des strings (in Festkomma)

9.10. Leistung der Versorgungsunterprogramme

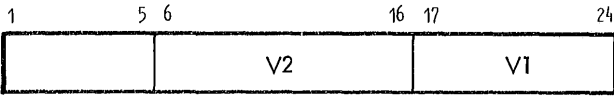
Für die wichtigsten Arten aktueller Parameter (A.P) soll die Leistung des Unterprogramms, das den aktuellen Parameter berechnet, angegeben werden.

- AP ist eine einfache oder indizierte Variable
 = Adresse der Variablen
<A> = Wert der Variablen
- AP ist eine Konstante oder ein Ausdruck
<A> = Wert
 = undefiniert
- AP ist der Name eines Feldes
 = Adresse des Informationsvektors
(siehe Informationsvektor eines Feldes)
- AP ist eine Marke, kein integer-label.
Sprung auf die Marke
- AP ist ein integer label
In diesem Fall führt das Versorgungsunterprogramm den Sprung nicht aus, sondern liefert in
<A> = Wert der Konstanten (in Gleitkomma)
 = Adresse der Marke
<H> = Basisadresse der Hierarchie des integer label

9.11. Aufbau der Zusatzversorgung

Die Zusatzversorgung (ZV) eines aktuellen Parameters enthält die Beschreibung des Parameters oder einen Verweis auf diese Beschreibung.

- ZV enthält einen Verweis



- V1 Indexadresse eines Elements des Globalvektors. $\langle V1 \rangle$ = Basisadresse der Hierarchie, in der die Beschreibung eingetragen ist.
- V2 Relativadresse des Halbwortes, in dem die Parameterbeschreibung steht.

Die Direktangabe der Parameterbeschreibung steht auf der Adresse

$$\langle \langle X11 \rangle + V1 \rangle + V2$$

Die Berechnung der Direktangabe erfolgt am besten mit Hilfe des Unterprogramms A&HVZV (siehe Beschreibung und Beispiel).

- ZV enthält die Parameterbeschreibung

*) siehe Zeichnung

- PS Prozedurspezifikation PS = 0 ... ALGOL PS = 1 ... FORTRAN
- NEL Zuweisungsbits, NEL = L bedeutet, daß der Parameter keine Wertzuweisung bekommen kann (z.B. eine Konstante)
- DA Direktangabe, DA = L wenn ZV die Parameterbeschreibung enthält
- DIM/PZ Dimension bzw. Parameterzahl für Felder und Prozeduren
- D D = L : DIM/PZ ist definiert D = 0 : DIM/PZ ist undefiniert
- ART-TYP Art-Typ-Beschreibung des aktuellen Parameters, die einzelnen Bits haben folgende Bedeutung
 - I integer
 - R real
 - B Boolean
 - T type - Größe (einfache Variable)
 - L label
 - A array
 - SW switch
 - ST string
 - P procedure

Beispiele:

Für die Konstante 1.2 steht in ZV

- NEL = L
- DA = L
- D = 0
- R = L
- T = L

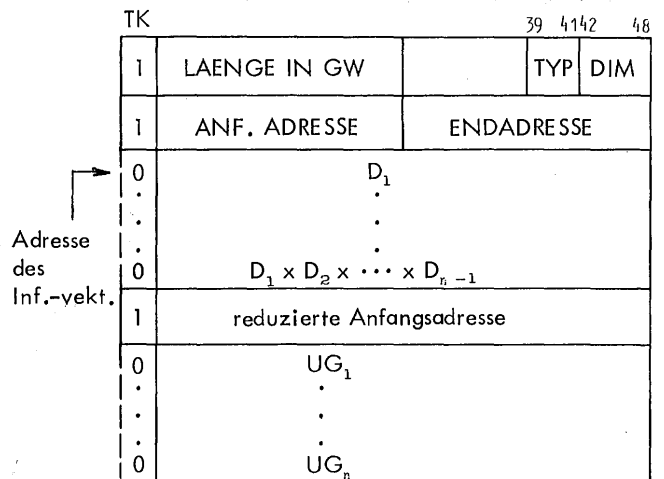
die übrigen Art-Typ-Bits sind gleich 0

Für den Namen einer boolean procedure mit 3 Parametern enthält ZV

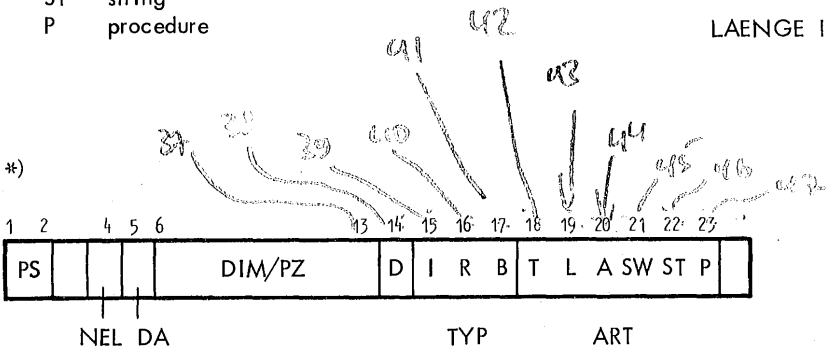
- NEL = L
- DA = L
- DIM/PZ = 3
- D = L
- B = L
- P = L

9.12 Informationsvektor eines Feldes

Der Informationsvektor eines n-dimensionalen Feldes umfaßt $2 \cdot n + 2$ Ganzworte, er enthält folgende Information:



- TYP L00 integer 010 real 001 boolean
- DIM Dimension des Feldes
- ANF.ADRESSE Adresse des ersten Feldelementes
- ENDADRESSE Adresse des letzten Feldelementes
- LAENGE IN GW = (ENDADR.-ANF.ADR+2)/2



D_i ($i=1,2,\dots,n-1$) Dimensionslänge
 UG_i ($i=1,2,\dots,n$) untere Grenze der i -ten Dimension
 reduzierte Anfangsadresse die Adresse des (evtl. fiktiven) Feldelementes A $[0, \dots, 0]$. Diese Größe dient als Basis für die Adreßrechnung.

Die Anordnung der Feldelemente erfolgt "spaltenweise" nach der Abbildungsfunktion

$$\rangle A[i_1, \dots, i_n] \langle = \text{Anf. adr.} + \delta_1 + D_1 * (\dots + D_{n-2} * (\delta_{n-1} + D_{n-1} * \delta_n) \dots)$$

mit $\delta_k = i_k - UG_k$

Mit Hilfe der reduzierten Anfangsadresse

$$\rangle A[0, \dots, 0] \langle = \text{Anf. adr.} - UG_1 + D_1 * (\dots + D_{n-2} * (-UG_{n-1} - D_{n-1} * UG_n) \dots)$$

erhält man

$$\rangle A[i_1, \dots, i_n] \langle = i_1 + i_2 * D_1 + \dots + i_n * D_1 * \dots * D_{n-1} + \rangle A[0, \dots, 0] \langle$$

Die vom Versorgungsunterprogramm gelieferte Adresse des Informationsvektors ist die Adresse des 3. Ganzwortes, für Felder der Dimension 1 steht dort die reduzierte Anfangsadresse.

Beispiel:

Bei der Deklaration des Feldes

`array a[-5: +5, 0.83: 2]`

wird - der aktuelle Freispeicherpegel sei '1820' - der folgende Informationsvektor generiert:

Adresse des Inf.-vekt.	'1820' =	1	'16'	'102'
		1	'182C'	'1856'
		0		11
		1		'1820'
		0		-5
		0		1

Bemerkung:

Der Informationsvektor eines Feldes und die Feldelemente werden gewöhnlich hintereinander im Freispeicher abgelegt. Für Felder, die in einer Common-Zone liegen, wird der Informationsvektor in der Adreßzone 2 des ALGOL-Objekts eingetragen.

Das Versorgungsunterprogramm liefert die Adresse '1824' als Adresse des Informationsvektors.

9.13. Bereitstellung von Haupt- und Zusatzversorgung A&HVZV

Zweck des Programms

Bereitstellung von Haupt- und Zusatzversorgung für aktuelle ALGOL-Parameter in ALGOL-Unterprogrammen, die in TAS geschrieben werden.

Aufbau

Programmiersprache: TAS

Programmform: Internes Unterprogramm

Handhabung

- Deklaration: In TAS-Programmen mit EXTERN A&HVZV,
- Aufruf: In TAS-Programmen SFBE (A&HVZV/A),
 Folgende Parameter müssen an A&HVZV übergeben werden:
 Register A:
 Relative Adresse der Haupt- und Zusatzversorgung eines aktuellen ALGOL-Parameters im Versorgungsblock.
 Indexzelle 11: Adresse des Versorgungsblocks
 Indexzelle 13: AUFHIER
 Indexzelle 16: AKTHIER
 Nach dem Rücksprung aus A&HVZV werden folgende Parameter übergeben:
 Register A:
 Haupt- und Zusatzversorgung des aktuellen ALGOL-Parameters
 Register Q, D, H: undefiniert
 Register B: Inhalt von AUFHIER
 Indexzelle 12: Inhalt von AUFHIER
 Indexzelle 0: Rücksprungadresse
 Beispiel:
 BA 2, SFBE (A&HVZV/A), bringt die Haupt- und Zusatzversorgung des 1. Parameters.

- Speicherbedarf: 17 Befehle, 2 Ganzwort Konstanten
 Die Indexzellen von 0-7 werden von A&HVZV nicht zerstört.

Arbeitsweise

Verfahren:
 Ist in der Zusatzversorgung eines aktuellen ALGOL-Parameters das Bit 5 nicht gesetzt (ZV enthält Verweis), so wird die aktuelle Zusatzversorgung über die Verweiskette bestimmt.

Gültigkeitsbereich:
 Für alle aktuellen ALGOL-Parameter gültig.

Genauigkeit: entfällt

Fehlerbehandlung:
 Keine Fehlerbehandlung. Es wird die richtige Besetzung aller Register und Indexzellen beim Aufruf von A&HVZV vorausgesetzt.

9.14. Beispiele

Die Prozedur DOPMULT (6 Parameter, die 3 doppelte
Zahlen repräsentieren) bewirkt die Multiplikation von 2
doppelte Zahlen.
P1 bis P4 sind Eingangsgrößen, das Resultat steht in P5, P6.

```

DOPMULT, = SEGM                                == ALGOL == DOPELTLANGE MULTIPLIKATION ==
EXTERN                                     A&FSP,
STRUKT (2),
EINGG (DOPMULT),
S&CA= CZONE V,
ABLAGS S&CA (V0),
S&C5= ASP 4/G,
AEND (V0),
BD S&C5,
R MCFU DL,
QCR 8,
R MCF DL,
TBC 16,
MAB BCI 14,
BA 15,
BAR 1,
SFBE

TXX 12 A&FSP/A),
MFU 11, 13,
T 2,
MF 16,
C 20,
TXX 12 13,
MFU 11,
T 4,
MF 16,
C 22,
TXX 12 13,
MFU 11,
T 6,
MF 16,
C 24,
TXX 12 13,
MFU 11,
T 8,
MF 16,
C 26,
TXX 12 13,
MFU 11,
T 10,
XC 28,
TXX 12 13,
MFU 11,
T 12,
XC 29,
MF 16,
BZ 20,
MAB DML 24,
E 28,
E 29,
TXR 16,
BA 3,
SFBE

MF 16, A&FSP/A),
BCI 14,
MAB SE 8,
ENDE,

```

Bild 9.6 Code - Prozedur DOPMULT

D

IAS - Handbuch

Die Code-Prozedur TAUSCH (2 Parameter, einfache Variable oder Felder) in Bild 9.6, bewirkt den Tausch der beiden Parameter.

TAUSCH.=	SEGM,	(TAUSCH),	
	EINGG		
	STRUKT (2),		
	EXTERN	A&FSP,	
	EXTERN	A&HVZV,	
S&CA=	CZONE	V,	
	ABLAG	S&CA	(VO),
S&C5=	ASP	4/G,	
S&C6=	ASP	4/G,	
	AEND	(VO),	
	BD	S&C5,	
	R	MCFU	DL,
	QCR	8,	
	R	MCF	DL,
	TBC	16,	
	MAB	BCI	14,
	BA	10,	
	BAR	1,	
	SFBE	AFSP,	
	MF	11,	
	BH	0,	
	ETA	'FF',	
	BAR	2,	
	SN	FEHLER	--PARAMETERZAHL = 2--
	BA	2,	
	SFBE	AHVZV,	
	SBIT	FELD	44A,
	SBIT	VAR	42A,
	S	FEHLER	--PARAMETER WEDER FELD NOCH EINFACHE VARIABLE--
VAR=	BA	4,	
	SFBE	AHVZV,	
	SBIT	2	42A,
	S	FEHLER	--PRUEFUNG PARAMETER VARIABLE--
	TXX	12	13,
	MFU	11,	
	T	2,	
	XC	0	--TAUSCH VON EINFACHEN VARIABLEN--
	TXX	12	13,
	MFU	11,	
	T	4,	
	XC	1,	
	E	BC	0,
	E	C	1,
	S	SCHLUSS,	
FELD=	BA	4,	
	SFBE	AHVZV,	
	SBIT	2	44A,
	S	FEHLER	--PRUEFUNG PARAMETER FELD--
	TXX	12	13,
	MFU	11,	
	T	2,	
	MAB	BZ	-4,
	R	B2	AL,
	SBA	1,	
	TRX	AN	0 -- -LAENGE1 +1 NACH X2 --
	R	TCB	QL,
	VBA	2,	
	XC	1	--AADR1 -2 NACH X1 --
	TXX	12	13,
	MFU	11,	
	T	4,	
	MAB	BZ	-4,
	R	B2	AL,
	SBA	1,	
	TRX	AN	2 -- -LAENGE2 +1 NACH X2 --
	VXX	0	2,
	SXN	FEHLER	--PRUEFEN LAENGE1 = LAENGE2 --
	R	TCB	QL,
	VBA	2,	
	XC	2	--AADR2 -2 NACH X2 --
	EZ	B	1,
	EZ	BC	2,
	E	C	1,
	SZX	-3	0 --TAUSCH VON FELDERN--

D

```

SCHLUSS= TXR      H      16,
          BA      3,
          SFBE   AFSP,
          MF      16,
          BCI    14,
          MABI   SE      8,
FEHLER=  B2      TX,
          VEL    SSUS,
          C      S&C6+2,
          XB     8,
          TBC    S&C6,
          XBA    2,
          TBC    S&C6+1,
          BH     (
                    '0'/HG,
                    S&C6/A),
          MF     16,
          BCI    14,
          MAB    MCF     11,
          SE     0,

TEXT=    ''TAUSCH, FALSCHER PARAMETER'',
          (''25''),
SSUS=    '020AAA'/2L,
TX=      TEXT/A,
AFSP=    A&FSP/A,
AHVZV=   A&HVZV/A,

          ENDE,

```

Bild 9.7 Code - Prozedur TAUSCH

10.1. Überblick

Der "Standardrahmen" erleichtert das Schreiben von weniger anspruchsvollen TAS-Programmen, indem er einige Standardvereinbarungen in Kraft setzt und Ein-Ausgabe-Routinen verfügbar macht. Im Abschnitt 6.6 findet man ein Beispiel für ein gerahmtes TAS-Programm, das zur ersten Orientierung dienen kann.

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

- im UEBERSETZE-Kommando darf nicht VARIANTE= stehen,
- das STARTE-Kommando muß Angaben für DUMP enthalten, z.B. DUMP=T-ALLES.

Für den Anschluß des Überwachers sind nur noch nötig:

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

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

SCHLUSS.=NULL 0,
```

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

- Die Protokollierung durch den TAS-Übersetzer darf nicht ausgeschaltet sein (durch den Befehl DRUCK 0 wäre sie ausgeschaltet).
- Im UEBERSETZE-Kommando muß mindestens PRØTØKØLL = Ø angegeben sein.
- Im MØNTIERE-Kommando muß mindestens PRØTØKØLL = A angegeben sein.

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

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

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

- Für Sonderfälle: Will man den Standardrahmen in mehreren Montageobjekten benutzen, so darf man nur dasjenige Montageobjekt mit der Spezifikation SPRACHE=TASR übersetzen, das als erstes auszuführen ist; die übrigen müssen mit SPRACHE=TAS (oder TASE) übersetzt werden und (spätestens) vor dem ersten Aufruf eines Makros des Standardrahmens die Pseudobefehle

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

enthalten(die EXTERN-Befehle heranschaffen).

10.2. Allgemeine Regeln für den Standardrahmen

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

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

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

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

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

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

BUE- ODER TK-ALARM BEIM EINTRITT IN RAHMEN gemeldet und der Operatorlauf abgebrochen.

Bei Alarm, SSR-Fehler, einem vom Standarddrahmen entdeckten Fehler und Ausführung einer vom Makro R&FEHLER (siehe Abschnitt 6.5.1) erzeugten Befehlsfolge geschieht folgendes:

- Gegebenenfalls wird der Überwacher ausgeschaltet.
- Fehlertext, der Art und Ort des Fehlers erkennen läßt, wird ausgedruckt.
- Gegebenenfalls werden gemäß der Spezifikation DUMP des STARTE-Kommandos Variable samt ihren Namen gedumt (z.B. bei TK1 als echter Bruch). Zu diesen Variablen gehören normalerweise auch die folgenden des Standardrahmens: R&EPUFFER, R&EPUFFERPØS, R&APUFFERPØS, R&VØRSCHUB, R&APUFFER, R&APUFFERVERLAENGERUNG, die insgesamt 226 Halbworte belegen, d.h. höchstens 2 Druckseiten liefern.
- Bei Software-Alarm (z.B. Überschreitung der im Abschnittskommando angegebenen Zeit- oder Druckseitenschranke) wird der Abschnitt abgebrochen, sonst, also u.a. bei SSR-Fehler oder Hardware-Alarm (z.B. Typenkennungs-Alarm, arithmetischer Alarm, Speicherschutzalarm, Überlauf des Registers U, ungültiger Befehlscode) wird der Operatorlauf beendet und zum nächsten Kommando übergegangen.

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

```
BEN=ASP O/B, R&LIES,
```

```
BEN=NULL O, R&LIES,
```

10.3. Das Makro R&RAHMEN

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

Aufruf: Zweckmäßigerweise ruft man das Makro R&RAHMEN implizit mit Hilfe der Kommando-Spezifikation SPRACHE=TASR auf (siehe Abschnitt 6.1).

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

Ein Befehl UNTPR 247, wird erzeugt.

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

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

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

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

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

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

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

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

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

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

10.4. Makros für die Ein- und Ausgabe

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

10.4.1. Das Makro R&LIES

Form des Aufrufs:

```
R&LIES[ (<16-Bit-Sprungadr. nach Stringende>)],
```


Länge des erzeugten Programmstücks: 2 Befehle.
 Voraussetzung: Bei Verwendung dieses Makros wird normalerweise im STARTE- (oder RECHNE-) Kommando ein DATEN-Fremdstring angegeben sein. Das Zeichen / der DATEN-Spezifikation muß entweder am Ende einer Zeile (=ausgenutzter Teil einer Lochkarte) stehen oder dem Zeichen / dürfen in derselben Zeile nur Blanks folgen. Nur die darauf folgenden Zeilen (möglicherweise 0 Zeilen) bis vor das nächste Fluchtsymbol gelten dann als Fremdstring.

Wirkung: Dieses Makro liest aus dem DATEN-Fremdstring des STARTE (oder RECHNE-) Kommandos ungeachtet einer evtl. Kartenummerierung die nächste Zeile in Form von Zentralcode-Oktaden (pro Halbwort eine Oktade und zwar rechtsbündig und mit NUL-Zeichen aufgefüllt) in den Eingabepuffer ab Halbwort R&EPUFFER und speichert im Register A die Anzahl der eingelesenen Oktaden. Der Eingabepuffer-Zeiger wird auf das oberste Halbwort des Eingabepuffers gerichtet. Die in den Eingabepuffer gelieferte Information kann dann beliebig weiter bearbeitet werden, z.B. durch Wortgruppen-transport oder mit den Makros von Abschnitt 6.4.3.

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

R&LIES FINDET WEDER DATEN NOCH SPRUNGADRESSE

und der Fehlerort ins Ablaufprotokoll gedruckt und der Operatorlauf abgebrochen (genauer in Abschnitt 6.2).

In einigen Fällen wird der Fehler

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

gemeldet und der Operatorlauf abgebrochen.

Erweiterung für Sonderfälle:

Daten können statt als DATEN-Fremdstring auch durch die Spezifikation

DATEI = 98-[(Datenbasisname).](Dateiname)

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

10.4.2. Der Eingabepuffer-Zeiger und die Makros R&EPOES und R&ESKIP

Der Standardrahmen führt einen Zeiger (R&EPUFFERPOES) ein, der (in Form absoluter Adressen) angibt, von welchem Halbwort des Eingabepuffers ab die nächsten Oktaden zu verarbeiten sind (pro Halbwort ist eine Oktade gespeichert). Der Zeiger zeigt nach Ausführung des Makros R&LIES auf das erste Halbwort des Eingabepuffers. Der Zeiger kann

durch das Makro R&EPOES auf eine anzugebende Position im Eingabepuffer gesetzt oder durch das Makro R&ESKIP um eine anzugebende Anzahl von Halbworten weitergerückt werden. Außerdem wird er durch die Makros von Abschnitt 6.4.3 weitergerückt, und zwar um die Anzahl von Halbworten, die das betreffende Makro verarbeitet.

10.4.2.1. Das Makro R&EPOES

Form des Aufrufs:

R&EPOES((L)), $1 \leq L \leq 80$

Länge der erzeugten Befehlsfolge: 2 Befehle
 Wirkung: Der Eingabepufferzeiger wird auf das L-te Halbwort des Eingabepuffers gerichtet (das die Adresse R&EPUFFER+L-1 hat).

10.4.2.2. Das Makro R&ESKIP

Form des Aufrufs:

R&ESKIP[(L)], $1 \leq L \leq 79$
 Vorbesetzung L = 1

Länge der erzeugten Befehlsfolge: 3 Befehle.
 Wirkung: Der Eingabepuffer-Zeiger wird um L Halbworten weitergerückt.

10.4.3. Makros zum Manipulieren von Eingabe-Information

Diese Makros dienen zum Herausschneiden, Umschlüsseln und Konvertieren von Information aus dem Eingabepuffer, die z.B. durch R&LIES dorthin kam.

Form des Aufrufs:

{	R&EOKT((L))	}	$1 \leq L \leq 6$	} Vorb. L=1
	R&ETET((L)[,<F-Adr.>])		$1 \leq L \leq 12$	
	R&EGANZ	}	$1 \leq L \leq 80$	
	R&EBRUCH		$2 \leq L \leq 80$	
R&EGLEIT		$4 \leq L \leq 80$		

Länge der erzeugten Befehlsfolge: Für R&EOKT 1 SU-Befehl und danach 1 NULL-Befehl, sonst 1 SU-Befehl und danach 2 weitere Befehle.

Bedeutung: Der Eingabepuffer-Zeiger gibt an, mit der wievielten Oktade des Eingabepuffers (pro Halbwort ist eine Oktade gespeichert und zwar rechtsbündig) die zu bearbeitende Oktadenfolge beginnt. Die Länge der zu bearbeitenden Oktadenfolge ist im Makroaufruf als Argument L angegeben. Es ist Sache des Benutzers, darauf zu achten, daß nur die durch R&LIES (oder sonstwie) definierte Information bearbeitet wird.

Im einzelnen geschieht folgendes: Bei Angabe von ØKT wird die Oktadenfolge gepackt, linksbündig in das Register A gebracht, mit NUL-Oktaden aufgefüllt und mit TK3 versehen.

Bei Angabe von TET wird die Oktadenfolge in eine Tetradenfolge umgeschlüsselt, rechtsbündig in das Register A gebracht, mit Tetraden O aufgefüllt und mit TK2 versehen. Zusatzregel: Führende Oktaden SP (blank, b) werden in Tetraden O umgeschlüsselt.

Bei der Angabe von GANZ muß die Oktadenfolge die Form

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

haben; sie wird (mit Hilfe des Befehls KDFR) in eine binäre Festpunktzahl konvertiert, und zwar im Register A rechtsbündig mit TK1.

Bei Angabe von BRUCH muß die Oktadenfolge die Form

$$b^0 - \infty [-b^0 - \infty] . \langle \text{Ziffer} \rangle^{1-13}$$

haben; sie wird in eine binäre Festpunktzahl konvertiert, und zwar im Register A linksbündig mit TK1. Der absolute maximale Fehler des Resultats hat die Größenordnung 2^{-43} .

Bei der Angabe von GLEIT muß die Oktadenfolge die Form

$$b^0 - \infty [-b^0 - \infty] . \langle \text{Ziffer} \rangle^{1-13} b^0 - \infty E b^0 - \infty [-b^0 - \infty] \langle \text{Ziffer} \rangle^{1-3}$$

haben; sie wird in eine binäre Gleitpunktzahl einfacher Genauigkeit konvertiert, und zwar im Register A mit TK0. Die Gleitpunktzahl hat einen maximalen absoluten Fehler von der Größenordnung

$$2^{-30} \cdot 16 \text{ Sedezimalexponent}$$

Nach der für die einzelnen Makros geschilderten Bearbeitung wird der Eingabepuffer-Zeiger um L Halbworte weitergerückt. Der Eingabepuffer bleibt unverändert.

Undurchführbare Anweisungen werden wie folgt behandelt: Bei unzulässigem Argument wird eine Fehlermeldung ausgegeben und der Operatorlauf beendet (genauer in Abschnitt 6.2). Bei falschen Eingabedaten wird ohne Fehlermeldung auf die im Makroaufruf evtl. angegebene 16-Bit-Fehleradresse gesprungen, anderenfalls eine der folgenden Fehlermeldungen gegeben

UNZULAESSIGES ZEICHEN
ZU GRÖSSE ZAHL
FEHLERHAFTE EINGABEDATEN

und der Operatorlauf abgebrochen.

10.4.4. Der Ausgabepuffer-Zeiger und die Makros R&APØS und R&ASKIP

Der Standardrahmen führt einen Zeiger (R&APUFFERPØS) ein, der (in Form absoluter Adressen) angibt, von welchem Halbwort des Ausgabepuffers ab die nächsten Oktaden zu

speichern sind (pro Halbwort wird eine Oktade gespeichert). Der Zeiger zeigt bei Start des Programms wie auch nach Ausführung des Makros R&DRUCKE auf das erste Halbwort des Ausgabepuffers. Der Zeiger kann durch das Makro R&APØS auf eine anzugebende Position im Ausgabepuffer gesetzt werden oder durch das Makro R&ASKIP um eine anzugebende Anzahl von Halbworten weitergerückt werden. Außerdem wird er durch die Makros von Abschnitt 6.4.5. weitergerückt, und zwar um die Anzahl von Halbworten, die zum Speichern der Oktadenfolge erforderlich ist, die das betreffende Makro erzeugt. Normalerweise ist pro Halbwort des Ausgabepuffers ein Druckzeichen (oder Leerzeichen) gespeichert; ist diese Voraussetzung in Sonderfällen nicht erfüllt, so besteht im allgemeinen kein einfacher Zusammenhang mehr zwischen dem Zeiger des Ausgabepuffers und der Position auf dem Ablaufprotokoll.

10.4.4.1. Das Makro R&APØS

Form des Aufrufs:

$$R\&APØS(\langle L \rangle), \quad 1 \leq L \leq 116$$

Länge der erzeugten Befehlsfolge: 2 Befehle.

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

10.4.4.2. Das Makro R&ASKIP

Form des Aufrufs:

$$R\&ASKIP[\langle L \rangle], \quad 1 \leq L \leq 116$$

Vorbesetzung L = 1

Länge der erzeugten Befehlsfolge: 3 Befehle.

Wirkung: Der Ausgabepuffer-Zeiger wird um L Halbworte weitergerückt.

10.4.5. Makros zum Zusammensetzen von Ausgabe-Information

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

Form des Aufrufs:

$$\left\{ \begin{array}{l} R\&ØKT \\ R\&ATET \\ R\&AGANZ \\ R\&ABRUCH \\ R\&AGLEIT \end{array} \right\} [\langle L \rangle], \quad \begin{array}{l} 1 \leq L \leq 6, \text{ Vorbes. } L = 6 \\ 1 \leq L \leq 12, \text{ Vorbes. } L = 12 \\ 1 \leq L \leq 14, \text{ Vorbes. } L = 14 \end{array}$$

Länge der erzeugten Befehlsfolge: Für R&AØKT, R&ATET und R&AGANZ 1 SU-Befehl und danach 1 NULL-Befehl, sonst 1 SU-Befehl.

Diese Makros erzeugen eine Oktadenfolge nach folgenden Regeln:

Bei Angabe von ØKT werden die L linkersten Oktaden aus dem Register A genommen. Die Typkennung im Register A bleibt unbeachtet. Die Zeichen werden ohne Prüfung auf Druckfähigkeit in den Ausgabepuffer übertragen (auf diese Weise können z.B. auch NUL-Oktaden in den Ausgabepuffer an Stellen kommen, an denen normalerweise Druckzeichen (oder Leerzeichen) stehen, so daß sich die vermutete Wirkung von R&APØS und R&ASKIP ändert).

Bei Angabe von TET werden die L rechtsletzten Tetraden aus dem Register A genommen und in Oktaden umgeschlüsselt. Die Typkennung im Register A bleibt unbeachtet.

Bei Angabe von GANZ wird aus dem Register A eine rechtsbündige Festpunktzahl genommen und in eine L-stellige Oktadenfolge der Form

{b|-}<Ziffer>^{L-1}

konvertiert; führende Nullen werden durch Leerzeichen ersetzt, ausgenommen in der rechtsletzten Stelle der Zahl. Die Typkennung im Register A bleibt unbeachtet. Die negative Null wird mit Minuszeichen gedruckt. Kann der Betrag der auszubehenden Zahl nicht mit L - 1 Dezimalstellen dargestellt werden, so wird die Fehlermeldung ZU GROSSE ZAHL gegeben und der Operatorlauf abgeschlossen.

Bei Angabe von BRUCH wird aus dem Register A eine linksbündige Festpunktzahl genommen und (mit Hilfe des Befehls KFLD) in eine 15-stellige Oktadenfolge der Form

{b|-}.<Ziffer>¹⁵

konvertiert. Die Typkennung im Register A bleibt unbeachtet. Die negative Null wird mit Minuszeichen gedruckt. Mögliche Fehlermeldung während der Übersetzung:

R&ABRUCH HAT FALSCHES ARGUMENT.

Der maximale, absolute Fehler der konvertierten Zahl ist von der Größenordnung 10^{-15} .

Bei Angabe von GLEIT wird aus dem Register A eine Gleitpunktzahl einfacher Genauigkeit genommen und in eine 19stellige Oktadenfolge der Form

{b|-}.<Ziffer>¹²E{b|-}<Ziffer>⁶

konvertiert. Die Typkennung im Register A bleibt unbeachtet. Mögliche Fehlermeldung während der Übersetzung:

R&AGLEIT HAT FALSCHES ARGUMENT.

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

$10^{-12} \cdot 10^{\text{Dezimal exponent}}$

Die so erzeugte Oktadenfolge wird in dasjenige Halbwort und die nachfolgenden gespeichert, auf das der Ausgabepuffer-Zeiger zeigt; gespeichert wird in gespreizter Form (d.h. NUL NUL Oktade

pro Halbwort), und zwar nur, soweit gemäß dem Ausgabepuffer-Zeiger noch Platz im Ausgabepuffer ist. Überschüssige Information geht verloren. Nach dem Abspeichern wird der Ausgabepuffer-Zeiger um L Halbwoorte (meist gleich der Anzahl der erzeugten druckbaren Oktaden) weitergerückt.

10.4.6. Das Makro R&DRUCKE

Form des Aufrufs:

R&DRUCKE,

Länge der erzeugten Befehlsfolge: 1 SU-Befehl. Wirkung: Dieses Makro druckt die im Ausgabepuffer vorliegende Folge von Zentralcode-Oktaden in das Ablaufprotokoll (bei Konsolbetrieb auch in das Konsolprotokoll). Der Puffer kann z.B. durch Wortgruppentransport oder durch die Makros von Abschnitt 6.4.5 gefüllt worden sein. Die Zeilenlänge beträgt 116 Zeichen (unabhängig vom Ausgabepuffer-Zeiger); darüber hinausgehende Zeichen gehen verloren. Der Papiervorschub vor dem Drucken richtet sich nach der linksersten Oktade in R&VØRSCHUB; diese Oktade kann man mit Hilfe des Makros R&SEITE oder R&VØRSCHUB wählen, wenn man vom Standard abweichen will; Standard ist der Vorschub auf die nächste Zeile und nach 63 Zeilen der Vorschub auf die nächste Seite. Nach dem Drucken werden alle Halbwoorte des Ausgabepuffers auf NUL NUL SP gesetzt, als Papiervorschub für einen späteren Aufruf von R&DRUCKE "nächste Zeile" vorgesehen und der Ausgabepuffer-Zeiger auf das erste Halbwort des Ausgabepuffers gerichtet.

Erweiterung für Sonderfälle:

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

DATEI = 99 - [<Datenbasisname>.]<Dateiname>

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

10.4.7. Das Makro R&SEITE

Form des Aufrufs:

R&SEITE,

Länge der erzeugten Befehlsfolge: 2 Befehle. Wirkung: Dieses Makro sorgt dafür, daß am Anfang des nächsten Aufrufs von R&DRUCKE des Ablaufprotokolls auf den nächsten Seitenanfang vorgeschoben wird.

10.4.8. Das Makro R&VØRSCHUB

Form des Aufrufs:

R&VØRSCHUB[(<i>)], $0 \leq i \leq 6$ Vorbes. $i = 1$

Wirkung: Dieses Makro sorgt dafür, daß am Anfang des nächsten Aufrufs von R&DRUCKE das Ablaufprotokoll um i Zeilen vorgeschoben wird. (Zusatz: Die Wirkung eines Vorschubs mit $i > 1$, welcher über den standardmäßigen 63-Zeilen-Raum einer Seite hinausreicht, ist nicht festgelegt.)

10.4.9. Das Makro R&BINAERDUMP

Form des Aufrufs:

R&BINAERDUMP (<Anfangsadresse>, <Endadresse>),

Hierbei sind Anfangs- und Endadressen 16-Bit- oder 22-Bit-Adressen, die als Ganzwortadressen gedeutet werden.

Länge der erzeugten Befehlsfolge: 2 Befehle.

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

Zusatzregel: Durch den Makroaufruf

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

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

R&EPUFFER. = ASP 80/G,
R&EPUFFERPØS. = ASP 1,
R&APUFFERPØS. = ASP 1,
R&VØRSCHUB. = ASP 2/G,
R&APUFFER. = ASP 116/G,

10.5. Makros zum Beenden der Programmausführung

10.5.1. Das Makro R&FEHLER

Form des Aufrufs:

R&FEHLER(<Oktadenkonstante>),

Hierbei darf die Oktadenfolge (die 4 Apostrophe nicht mitgezählt) höchstens 100 Zeichen lang sein.

Wirkung:

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

TEST*EIN, TEST*AUS,

ausgeführt.

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

Gegebenenfalls folgt ein Variablen-Dump.

Der Operatorlauf wird beendet und das nächste Kommando bearbeitet.

Beispiel:

Der Aufruf R&FEHLER("UNERWARTETER FALL"), kann z.B. folgende Fehlerausschrift liefern:

UNERWARTETER FALL FEHLERØRT: 00142C

10.5.2. Das Makro R&ENDE

Form des Aufrufs:

R&ENDE,

Länge des erzeugten Programmstückes: 1 Befehl.

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

Der Operatorlauf wird beendet und das nächste Kommando bearbeitet.

10.5.3. Die Adresse R&ENDE

Unter dem globalen Namen R&ENDE ist die 16-Bit-Adresse derselben Befehlsfolge verfügbar, die auch durch das Makro R&ENDE (siehe Abschnitt 6.5.2.) aufgerufen wird.

Beispiel:

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

10.5.4. Die Adresse R&SSRFEHLER

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

Aufgrund der Registerinhalte wird eine Fehler-
nachricht gegeben, die etwa so aussieht:

```
FEHLER NR. 56 BEI SSR 253 10
AUFTRUF VON ADR. 003650
<Q> = 3 000000000000
<H> = 2 0000000001F
```

Gegebenenfalls folgt ein Variablendump.

Der Operatorlauf wird beendet und das nächste
Kommando bearbeitet.

D

10.6. Beispiel

Bei Verwendung der Spezifikation SPRACHE=
TASR im UEBERSETZE-Kommando bilden die folgen-
den Zeilen bereits ein vollständiges TAS-Pro-
gramm:

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

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

Die Verfügbarkeit kann nur durch LDEF-Befehle oder LDEFAB-Befehle (siehe Abschnitt 8.4.2) enden.

11.1. Organisation einer Bibliothek als Datei

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

Genauer: Eine Makrobibliothek ist eine RAN-Datei mit dem Satzbau U200W, für die außer den Regeln der Datenbasis noch folgendes gilt:

- Jeder Satz der Datei (außer dem Satz Nr. 1) enthält ein Makro im Sinne der TAS-Sprache. Der Satz Nr. 1 enthält ein (hier nicht weiter beschriebenes) Inhaltsverzeichnis, das für n Makros aus 1+6n Ganzworten besteht.
- Die Makronamen müssen innerhalb der betreffenden Makrobibliothek eindeutig sein.
- Die Makronamen dürfen als Sonderzeichen nur & aber nicht * enthalten.

11.2. Der Begriff "Verfügbarkeit eines Makros"

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

- a durch eine im TAS-Quellenprogramm stehende (mit DEF beginnende) Makrodefinition, ausgenommen durch eine Definition innerhalb einer anderen Makrodefinition, oder
- b durch einen HDEF-Befehl (siehe Abschnitt 8.4.1) oder
- c durch den Aufruf eines (verfügbaren) Makros, bei dessen Ausführung die Bedingungen a oder b in Erfüllung gehen.

Für die momentan verfügbaren Makros ist bekannt

- in welcher Reihenfolge sie verfügbar wurden,
- ob sie durch einen Sammeltransport (siehe Abschnitt 8.4.1) einer ganzen Makrobibliothek verfügbar wurden,
- gegebenenfalls von welcher Bibliothek der Sammeltransport ausging.

11.3. Aufruf eines Makros

Nur verfügbare Makros können aufgerufen werden. Falls mehrere Makros mit gleichen Namen verfügbar sind, ist das zuletzt verfügbar gewordene gemeint.

Anmerkung: Beachte, daß Makroaufrufe nicht benannt sein dürfen.

11.4. Pseudobefehle zum Beginnen und Beenden der Verfügbarkeit von Makros

Anwendungshinweis:

Mit den folgenden Pseudobefehlen hat man es in der Hand, die Verfügbarkeitsbereiche der Makros beliebig festzulegen. Will man extrem an Speicherplatz während des Übersetzens sparen, so wird man jedes Makro erst unmittelbar vor seinem Aufruf verfügbar machen und schon unmittelbar danach seine Verfügbarkeit wieder beenden. Steht hingegen genügend viel Speicherplatz zur Verfügung, so wird man alle gewünschten Bibliotheksmakros schon am Programm-Anfang verfügbar machen und auf Befehle zum Beenden ihrer Verfügbarkeit ganz verzichten.

11.4.1. Der HDEF-Befehl (Hole Definition)

Form:

```
HDEF [L<Datenbasisname>]
      (<Dateiname> [,<Makroname>]0-20),
```

Anmerkung zur Syntax:

Bekanntlich darf der Datenbasisname höchstens 6 Zeichen lang sein und der Dateiname höchstens 12 Zeichen. Der Dateiname darf nicht, wie in der Kommandosprache, eine Generationsnummer oder Versionsnummer enthalten.

Bedeutung: Die angegebene Datei (die dem Betriebssystem bereits bekannt sein muß) wird zur Bearbeitung eröffnet. Danach werden, falls eine oder mehrere Makronamen angegeben sind, die entsprechenden Makros aus der Datei geholt und in der Reihenfolge ihrer Satznummerierung verfügbar gemacht, anderenfalls (durch "Sammeltransport") alle Makros der Datei (in der Reihenfolge ihrer Satznummerierung). Danach wird die Dateibearbeitung beendet.

11.4.2. Der LDEF-Befehl und der LDEFAB-Befehl (Lösche Definition bzw. Lösche Definition ab)

In TAS stehen zum Löschen von Makrodefinitionen folgende Pseudobefehle zur Verfügung:

```
{ LDEF[_<Datenbasisname>]( <Dateiname> )  
  LDEFAB[_<Datenbasisname>]( <Dateiname> )  
  LDEF_<Makroname> } ,
```

Bedeutung der ungeklammerten Form mit LDEF:

Unter allen verfügbaren Makros mit dem angegebenen Makronamen (ungeachtet evtl. Sammeltransport-Vermerke) endet die Verfügbarkeit des zuletzt verfügbar gewordenen Makros.

Bedeutung der geklammerten Form mit LDEF:

Die Verfügbarkeit aller aus dem letzten Sammeltransport der angegebenen Makrobibliothek noch verfügbaren Makros wird beendet. Zugleich wird auch der Vermerk über den Sammeltransport gelöscht. (Im Extremfall ist die Verfügbarkeit von jedem Makro dieses Sammeltransportes schon vorher einzeln beendet worden, so daß nun nur noch der Sammeltransport-Vermerk zu löschen ist.)

Bedeutung der Form mit LDEFAB:

Die Wirkung gleicht der Wirkung des LDEF-Befehls; außerdem werden noch für alle später verfügbar gewordenen Makros die Verfügbarkeit beendet und die Sammeltransport-Vermerke gelöscht.

Anmerkungen:

- Hinsichtlich des Zeitaufwandes ist der LDEFAB-Befehl günstiger als der LDEF-Befehl.
- Obige Befehle löschen Makrodefinitionen nur im Kernspeicher, nicht auch in der Makrobibliothek.

11.5. Kommandos für Manipulationen mit Makrobibliotheken

Damit die HDEF-Befehle ausgeführt werden können, müssen die betreffenden Makrobibliotheken dem Betriebssystem bekannt sein. Solange das Betriebssystem noch keine langfristige private Datenhaltung zuläßt, wird eine Makrobibliothek entweder im jeweiligen Abschnitt als externe Spuldatei in die private (Standard-) Datenbasis &STDB eingeschleust (mit dem VERLAGERE-Kommando) oder sie ist in der öffentlichen Datenbasis &DEFDB des Prozesses vorhanden.

Zum Aufbauen, Ändern und Drucken einer Makrobibliothek sind spezielle Kommandos verfügbar (siehe Kommando-Handbuch unter MEINTRAGE usw.). Man beachte, daß die Dateinamen von Makrobibliotheken weder eine Generationsnummer noch eine Versionsnummer enthalten dürfen.

TELEFUNKEN
COMPUTER

E

TAS - HANDBUCH

Übersetzen, Testen

INHALT

ÜBERSETZEN, TESTEN



4. UEBERSETZE-KOMMANDO 4 - 1 =====

4.1.	Quelle	4 - 1
4.2.	Sprache	4 - 1
4.3.	Numerierung	4 - 1
4.4.	Name des Montageobjekts	4 - 1
4.5.	Variante	4 - 1
4.6.	Protokoll	4 - 1
4.7.	Dynamische Kontrollen	4 - 1
4.8.	Trace	4 - 1
4.9.	Maintenance-Nummer	4 - 1
4.10.	Kontrollereignisse	4 - 1
4.11.	Transfer	4 - 1
4.12.	Version	4 - 1

5. DAS TAS-PROTOKOLL 5 - 1 =====

<u>5.1.</u>	<u>Der Protokollrahmen</u>	5 - 1
<u>5.2.</u>	<u>Das Quelleprotokoll</u>	5 - 1
5.2.1.	Unterdrückung des Quelleprotokolls	5 - 1
5.2.2.	Seitenvorschub und Kopfzeile	5 - 1
5.2.3.	Die Informationseinheiten	5 - 1
5.2.3.1.	Numerierung	5 - 1
5.2.3.2.	Objektcode	5 - 1
5.2.3.3.	Quelle	5 - 2
5.2.3.4.	Ganzwortliterale	5 - 2
<u>5.3.</u>	<u>Abbruch der Übersetzung in Fehlerfällen</u>	5 - 2
5.3.1.	Alarmer	5 - 2
5.3.2.	Nicht erbrachte Diestleistungen	5 - 2
5.3.3.	Besondere Fälle	5 - 2

7. DER ÜBERWACHER (TEST*-BEFEHLE) 7 - 1 =====

7.1.	Die Testbefehle	7 - 1
7.1.1.	TEST*EIN-Befehl	7 - 1
7.1.2.	TEST*AUS-Befehl	7 - 1
7.1.3.	TEST*SR-Befehl	7 - 1
7.1.4.	TEST*SD-Befehl	7 - 2
7.1.5.	TEST*FR-Befehl	7 - 2
7.1.6.	TEST*FD-Befehl	7 - 2
7.1.7.	TEST*CR-Befehl	7 - 2
7.1.8.	TEST*XR-Befehl	7 - 2
7.1.9.	TEST*BR-Befehl	7 - 2
7.1.10.	TEST*SPRUNG-Befehl	7 - 2
7.1.11.	TEST*4XR-Befehl	7 - 2
7.1.12.	TEST*FØRT-Befehl	7 - 3
7.1.13.	TEST*SSATZ-Befehl	7 - 3
7.2.	Allgemeines zu den Testbefehlen	7 - 3

E

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

4.1. Quelle

Keine weiteren Erklärungen.

4.2. Sprache

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

Zur Angabe von TASR siehe die Beschreibung des Rahmenprogramms in Abschnitt 6.

4.3. Numerierung

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

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

4.4. Name des Montageobjekts

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

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

4.5. Variante

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

4.6. Protokoll

Das Übersetzungsprotokoll ist in Abschnitt 5 beschrieben.

4.7. Dynamische Kontrollen

Entfällt bei TAS.

4.8. Trace

Im Falle "undefiniert" ignoriert der Assembler jeden Testbefehl in der Quelle und außerdem die auf ihn folgende Informationseinheit, falls es sich um einen Testbefehl mit Versorgung handelt. Die Testbefehle sind in Abschnitt 7 beschrieben.

4.9. Maintenance-Nummer

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

4.10. Kontrollereignisse

Entfällt bei TAS.

4.11. Transfer

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

4.12. Version

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

Wenn die Konstante eine Liste ist, muß zunächst ein FØRM-Befehl gegeben werden, bevor auf die Werte in dieser Liste zugegriffen werden kann. Siehe dazu Beschreibung und Beispiele in Abschnitt 3.9. sowie Abschnitt 3.1.2., letzter Absatz.

5.1. Der Protokollrahmen

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

Nach Abschluß der Protokollierung der Quelle werden folgende Angaben gemacht:

- Fehlermeldungen, die Kontaktnamen und Gebietsdeklarationen betreffen (falls Fehler vorliegen).
- Der Indexpegel, d.i. die höchste einem (symbolischen) Indexnamen zugewiesene Adresse; wird als Dezimalzahl ausgegeben.
- Falls durch das UEBERSETZE-Kommando das Protokoll unterdrückt wurde, erscheint (zur Kontrolle) die Anzahl der Informationseinheiten.
- Es werden die Seiten des Protokolls angezeigt, auf denen Fehlermeldungen zu finden sind (falls Fehler vorliegen); Bezugspunkt ist dabei die Seitennummer in der TAS-Kopfzeile, nicht die der Systemkopfzeile.
- Nach Ablage des Montageobjekts wird der in die Datei &MO eingetragene Montageobjektname ausgedruckt, gegebenenfalls mit der Meldung, daß ein gleichbenanntes älteres Montageobjekt gestrichen wurde. (Dieser Punkt entfällt, falls nur Syntaxprüfungen verlangt wurde.)
- Falls gewünscht, wird nun das Adreßbuch ausgegeben, wobei zuerst die globalen Namen erscheinen, dann segmentweise unter Angabe des Segmentnamens (falls vorhanden) die lokalen Namen. Wurden zusätzlich Referenzen gewünscht, so erhält man pro Zeile ein Adreßbuchelement, gefolgt von der Liste der Referenzen, die sich auf die Numerierung der Protokollzeilen beziehen; im anderen Fall werden vier Elemente pro Zeile gedruckt.

Format der Adreßbuchelemente (je 28 Druckspalten):

- Spalte 1: Ablagebereich, dabei bedeutet
E Externer Name
I Indexname
G gleichgesetzter Name
(K, V, B, D wie üblich)
- Spalten 2,3: Zonennummer (hexadekadisch),
entfällt bei I und G; bei E
handelt es sich um eine Pseudozonennummer (für die Montage).
- Spalten 4-9: Zonenrelative Adresse (hexadekadisch) entfällt bei G,
ist bei E stets 0 und bei I
eine "absolute" Adresse.
- Spalten 13-24: Name (linksbündig); bei Namen von mehr als 12 Zeichen
Länge werden nur die ersten
zwölf Zeichen gedruckt.

Anmerkung: Das referenzlose Adreßbuch wird hauptsächlich zur Auswertung von Dumps und

(unter Einbeziehung des Montageprotokolls) zur Herstellung von Korrekturen (in interner Form) benutzt.

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

5.2. Das Quelleprotokoll5.2.1. Unterdrückung des Quelleprotokolls

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

5.2.2. Seitenvorschub und Kopfzeile

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

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

5.2.3. Die Informationseinheiten5.2.3.1. Numerierung

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

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

5.2.3.2. Objektcode (falls verlangt)

- Die Zonennummer (erscheint nur bei Änderung der Ablagezone).
- Die zonenrelative Adresse (hexadekadisch) bezieht sich auf die zuletzt erschienene Zonennummer.

- Die Typenkennung; das Zeichen - bedeutet, daß der (Halbwort-) Wert keine eigene Typenkennung erhält (Halbworte auf ungerader Adresse).
- Die Werte:
Bei Ganzworten wird das entstandene Bitmuster in Form von 12 Tetraden gedruckt.

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

- 0 Der Halbwortwert wird bei der Montage nicht verändert.
- 1 (bei Befehlen) Auf den Adreßteil (Bits 9 bis 24) wird die Anfangsadresse der angegebenen Zone addiert (modulo 2^{16}).
- 2 (bei Adreßkonstanten) Auf das Feld der Bits 3 bis 24 wird die Anfangsadresse der Zone addiert (modulo 2^{23}).
- 4 (bei Oktadenadreßkonstanten) Auf den Halbwortwert wird das Dreifache der Anfangsadresse der angegebenen Zone addiert (modulo 2^{24}).

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

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

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

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

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

5.2.3.3. Quelle

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

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

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

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

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

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

5.2.3.4. Ganzwortliterale

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

5.3. Abbruch der Übersetzung in Fehlerfällen

5.3.1. Alarme (außer Ereignisalarmen)

Der Alarmkeller wird in Standardform ausgedruckt.

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

Der Variablenbereich des Assemblers wird gedumpt.

Anmerkung: Derartige Protokolle sind N31/P61 zuzustellen oder zu melden.

5.3.2. Nicht erbrachte Dienstleistungen

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

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

5.3.3. Besondere Fälle

In besonderen Fällen wie Listenüberläufe oder Erkennen "unmöglicher" Übersetzungszustände u.a.m. wird die Übersetzung mit dem Text "TAS-Abbruch" vorzeitig beendet. Hier gilt ebenfalls das unter 5.3.1. und der Anmerkung gesagte.

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

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

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

Die in einem Quellenprogramm enthaltenen Testbefehle (einschließlich der meist folgenden Adessenkonstanten bzw. Nullbefehle) werden vom Assembler genau dann überlesen, wenn die Spezifikation TRACE im UEBERSETZE-Kommando nicht definiert ist (siehe Abschnitt 4.8).

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

7.1. Die Testbefehle

Es gibt folgende Testbefehle

TEST*EIN	<u>E</u> inschalten des Überwachers
TEST*AUS	<u>A</u> usschalten des Überwachers
TEST*SR	Drucke <u>s</u> ofort die <u>R</u> egisterstände der als nächstes ausgeführten n Befehle
TEST*SD	Gib <u>s</u> ofort einen <u>D</u> ump der angegebenen Speicher- und Indexbereiche
TEST*FR	Falls der Befehlszähler in einem der angegebenen Bereiche ist, drucke die <u>R</u> egisterstände
TEST*FD	Gib bei Erreichen eines Befehlszählerstandes einen <u>D</u> ump der angegebenen Speicher- und Indexbereiche
TEST*CR	Falls ein Wort aus einem der angegebenen Speicherbereiche angesprochen wird, drucke die <u>R</u> egisterstände
TEST*XR	Falls eine <u>I</u> ndexzelle aus einem der angegebenen Index- oder Speicherbereiche angesprochen wird, drucke die <u>R</u> egisterstände
TEST*BR	Falls der Befehlscode einer der angegebenen ist, drucke die <u>R</u> egisterstände

TEST*SPRUNG Falls Sprungbefehl, drucke die Registerstände

TEST*4XR Simuliere die 4 Indexregister

TEST*FORT Fortsetzen nach einem SSR-Befehl

TEST*SSATZ Angaben der StarSatzadresse (für Seitenschranke)

7.1.1. TEST*EIN-Befehl

```
<TEST*EIN-Befehl> ::= TEST*EIN
```

Der TEST*EIN-Befehl bewirkt das dynamische Einschalten des Überwachers und gleichzeitigen Druck der Registerstände. Der eingeschaltete Überwacher überwacht alle auszuführende TR 440-Befehle und nimmt weitere Testbefehle entgegen.

7.1.2. TEST*AUS-Befehl

```
<TEST*AUS-Befehl> ::= TEST*AUS
```

Der TEST*AUS-Befehl bewirkt sofortiges Ausschalten des Überwachers und gleichzeitigen Druck der Registerstände. Vom Überwacher angesammelte Druckinformation wird ausgegeben. Im ausgeschalteten Zustand des Überwachers sind alle Testbefehle außer dem TEST*EIN und dem TEST*SSATZ ohne Wirkung.

7.1.3. TEST*SR-Befehl

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

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

7.1.4. TEST*SD-Befehl

```
<TEST*SD-Befehl> ::= TEST*SD,  
                    <V-Block-Adr. >/AB
```

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

7.1.5. TEST*FR-Befehl

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

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

7.1.6. TEST*FD-Befehl

```
<TEST*FD-Befehl> ::= TEST*FD,  
                    <V-Block-Adr. >/AB
```

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

7.1.7. TEST*CR-Befehl

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

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

7.1.8. TEST*XR-Befehl

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

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

7.1.9. TEST*BR-Befehl

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

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

7.1.10. TEST*SPRUNG-Befehl

```
<Test*SPRUNG-Befehl> ::= TEST*SPRUNG
```

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

7.1.11. TEST*4XR-Befehl

```
<TEST*4XR-Befehl> ::= TEST*4XR
```

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

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

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

7.1.12. TEST*FØRT-Befehl (Überwachung des SSR-Befehls)

⟨TEST*FØRT-Befehl⟩ ::= TEST*FØRT

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

7.1.13. TEST*SSATZ-Befehl

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

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

Ist die Seitenschranke erreicht, so wird der Überwacher ausgeschaltet.

Ist die Anfangsadresse des Startsatzes unbekannt, so kann man die Adresse eines simulierten Startsatzes, in dem nur das Drittelwort mit der Relativadresse 6 relevant ist (gibt die Seitenschranke an), mitgeben.

7.2. Allgemeines zu den Textbefehlen

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

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

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

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

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

TELEFUNKEN
COMPUTER

G

TAS - HANDBUCH

Konstanten, Pseudobefehle

INHALT



KONSTANTEN, PSEUDOBEFEHLE

<u>1.</u>	<u>KONSTANTEN</u>	1 - 1
1.1.	Gleitkommakonstante einfacher Genauigkeit	1 - 1
1.2.	Gleitkommakonstante doppelter Genauigkeit	1 - 2
1.3.	Festkommakonstante	1 - 3
1.4.	Tetradenkonstante	1 - 4
1.5.	Oktadenkonstante	1 - 5
1.6.	Textkonstante	1 - 6
1.7.	Adressenkonstante	1 - 7
1.8.	Bitfeldkonstante	1 - 8
1.9.	Konstante im Befehlsformat	1 - 8
<u>2.</u>	<u>AUFBAU VON QUELLENPROGRAMMEN</u>	2 - 1
2.1.	Programmanfang	2 - 1
2.2.	Programmende	2 - 1
2.3.	Segmente	2 - 1
2.4.	Voreinstellung für Operatoren	2 - 1
<u>3.</u>	<u>VEREINBARUNG VON SPEICHERBEREICHEN</u> ..	3 - 1
3.1.	Arbeitsspeicher	3 - 1
3.2.	Datenspeicher	3 - 1
<u>4.</u>	<u>VEREINBARUNG VON ADRESSENZONEN UND GEBIETEN</u>	4 - 1
4.1.	Adressenzone	4 - 1
4.2.	Commonzone	4 - 1
4.3.	Freihaltezone	4 - 2
4.4.	Gebiet	4 - 2
4.5.	Lücke	4 - 3
<u>5.</u>	<u>STEUERUNG DER ABLAGE</u>	5 - 1
5.1.	Verteilung der Informationseinheiten auf die Adressenzonen	5 - 1
5.2.	Anordnung von Adressenzonen in Gebieten	5 - 2
5.3.	Anordnung von Gebieten im Adressenraum	5 - 3
<u>6.</u>	<u>QUERBEZÜGE ZWISCHEN QUELLENPROGRAMMEN</u>	6 - 1
6.1.	Eingang	6 - 1
6.2.	Externbezug	6 - 2
<u>7.</u>	<u>ZUORDNUNGEN</u>	7 - 1
7.1.	Zuordnung der Indexnamen	7 - 1
7.2.	Vereinbarung von Oktadennamen	7 - 1
7.3.	Gleichsetzung	7 - 2
<u>8.</u>	<u>UMSCHALTEN DES PROTOKOLLDRUCKS</u>	8 - 1
<u>9.</u>	<u>MAKROS UND WIEDERHOLUNGEN</u>	9 - 1
9.1.	Makrodefinition	9 - 1
9.2.	Makroaufruf	9 - 1
9.3.	Parameter	9 - 1
9.4.	Versionen	9 - 4
9.5.	Wiederholungen	9 - 5

INHALT

KONSTANTEN, PSEUDOBEFEHLE

<u>1.</u>	<u>KONSTANTEN</u>	1 - 1
1.1.	Gleitkommakonstante einfacher Genauigkeit	1 - 1
1.2.	Gleitkommakonstante doppelter Genauigkeit	1 - 2
1.3.	Festkommakonstante	1 - 3
1.4.	Tetradenkonstante	1 - 4
1.5.	Oktadenkonstante	1 - 5
1.6.	Textkonstante	1 - 6
1.7.	Adressenkonstante	1 - 7
1.8.	Bitfeldkonstante	1 - 8
1.9.	Konstante im Befehlsformat	1 - 8
<u>2.</u>	<u>AUFBAU VON QUELLENPROGRAMMEN</u>	2 - 1
2.1.	Programmanfang	2 - 1
2.2.	Programmende	2 - 1
2.3.	Segmente	2 - 1
2.4.	Voreinstellung für Operatoren	2 - 1
<u>3.</u>	<u>VEREINBARUNG VON SPEICHERBEREICHEN</u> ..	3 - 1
3.1.	Arbeitsspeicher	3 - 1
3.2.	Datenspeicher	3 - 1
<u>4.</u>	<u>VEREINBARUNG VON ADRESSENZONEN UND GEBIETEN</u>	4 - 1
4.1.	Adressenzone	4 - 1
4.2.	Commonzone	4 - 2
4.3.	Freihaltezone	4 - 2
4.4.	Gebiet	4 - 2
4.5.	Lücke	4 - 3
<u>5.</u>	<u>STEUERUNG DER ABLAGE</u>	5 - 1
5.1.	Verteilung der Informationseinheiten auf die Adressen- zonen	5 - 1
5.2.	Anordnung von Adressenzonen in Gebieten	5 - 2
5.3.	Anordnung von Gebieten im Adressenraum	5 - 3
<u>6.</u>	<u>QUERBEZÜGE ZWISCHEN QUELLEN-PROGRAMMEN</u>	6 - 1
6.1.	Eingang	6 - 1
6.2.	Externbezug	6 - 1
<u>7.</u>	<u>ZUORDNUNGEN</u>	7 - 1
7.1.	Zuordnung der Indexnamen	7 - 1
7.2.	Vereinbarung von Oktadennamen	7 - 1
7.3.	Gleichsetzung	7 - 2
7.4.	Codeumsteuerung für Oktadenfolgen	7 - 2
7.5.	Oktadenadressierung	7 - 2
<u>8.</u>	<u>STEUERUNG DES PROTOKOLLDRUCKS</u>	8 - 1
8.1.	Umschalten des Protokoll- drucks	8 - 1
8.2.	Einfügen von Leerzeilen	8 - 1
8.3.	Unterdrückung "VERBOTENER BEZUG"	8 - 1
8.4.	Oktadenstring als Kommentar	8 - 2

G

9.	MAKROS UND WIEDERHOLUNGEN	9 - 1
9.1.	Makrodefinition	9 - 1
9.2.	Makroaufruf	9 - 1
9.3.	Parameter	9 - 1
9.4.	Versionen	9 - 4
9.5.	Wiederholungen	9 - 5

1. KONSTANTEN

Mit Hilfe der Konstanten-Schreibweise ist es möglich, ein Wort, Teile eines Wortes und mehrere Wörter mit einem bestimmten Wert zu besetzen. Die Schreibweise der Konstanten ist so gewählt, daß sie sich eindeutig von Befehlen und Pseudobefehlen unterscheiden. Damit können sie gemischt mit diesen niedergeschrieben werden.

Bei Konstanten unterscheidet man zwischen

- Gleitkommakonstanten einfacher Genauigkeit,
- Gleitkommakonstanten doppelter Genauigkeit,
- Festkommakonstanten,
- Tetradenkonstanten,
- Oktadenkonstanten,
- Textkonstanten,
- Adressenkonstanten und
- Bitfeldkonstanten.

Die Konstanten werden vom TAS-Assembler je nach Art und Spezifikation in Halbwörter, Ganzwörter oder Folgen von Ganzwörtern übersetzt und mit einer den jeweiligen Typ der Konstanten charakterisierenden Typenkennung versehen.

Normalerweise erfolgt die Ablage von Konstanten in einem schreibgeschützten und direkt adressierbaren Speicherbereich, dem sogenannten Konstantenbereich (K-Bereich).

Vom Normalfall abweichende Eigenschaften der Konstanten müssen durch entsprechende explizite Spezifikationen angegeben werden. Die Spezifikationen sind im Abschnitt "Aufbau", 5.5. näher erläutert.

Bei Gleitkomma- und Festkommakonstanten können innerhalb einer Dezimalzahl beliebig Zwischenräume eingeschoben werden. Sie dienen nur dazu, eine übersichtliche Gliederung zu erhalten und werden nicht übersetzt.

Um mit Sicherheit zu erreichen, daß ein Halbwort die gewünschte Typenkennung erhält, gibt man gleichzeitig mit einer expliziten Typenkennungsspezifikation die Spezifikation G (gerade Adresse = linkes Halbwort) an.

Beispiel: 100/H1G, wird als linkes Halbwort abgelegt und erhält mit Bestimmtheit die Typenkennung 1, da das folgende rechte Halbwort eine abweichende explizite Typenkennungsspezifikation haben kann.

100/H1U, erhält nicht mit Bestimmtheit die Typenkennung 1, da ungerade Adresse = rechtes Halbwort gewünscht.

1.1. Gleitkommakonstante einfacher Genauigkeit

Unter einer Gleitkommakonstanten einfacher Genauigkeit versteht man eine Gleitkommazahl, die ihrerseits aus einem Mantissteil und einem Exponententeil besteht. Die Mantisse kann eine ganze Zahl, ein echter Bruch oder ein unechter Bruch sein. Der Exponent ist immer eine ganze Zahl.

Gleitkommakonstanten einfacher Genauigkeit haben die Form

mEe , m: Mantisse, ganze Zahl oder Bruch;
 e: Exponent, ganze Zahl;
 dezimal zur Basis 10;
 (kann entfallen)

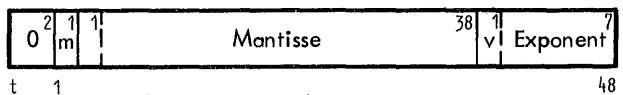
Es wird ein Zahlenbereich von 10^{-155} bis 10^{+152} erfaßt. Die Genauigkeit beträgt 10 Dezimalstellen.

Ein Dezimalkomma ist stets als Punkt zu schreiben.

Intern wird eine Gleitkommakonstante einfacher Genauigkeit im Normalfall (d.h. ohne besondere explizite Spezifikation) mit Typenkennung 0 in ein TR 440-Ganzwort mit Gleitkommastruktur umgewandelt. Die Ablage erfolgt ohne Markierung in dem Konstantenbereich (K-Bereich).

Typenkennung bei zwei Halbwortkonstanten

Da im Speicher nur Ganzwörter mit einer Typenkennung versehen werden, wird beim Abspeichern von zwei Halbwortkonstanten mit unterschiedlicher Typenkennung in einem Ganzwort immer die Typenkennung des linken Halbwortes wirksam. Das gilt auch für den Fall, daß das rechte Halbwort eine explizite Typenkennungsspezifikation besitzt.



m = Marke
 v = Vorzeichen

Bild 1.1 Interne Darstellung einer Gleitkommazahl einfacher Genauigkeit



TAS - Handbuch

Mai 70

Abweichungen vom Normalfall sind bei Gleitkommakonstanten mit Hilfe der Spezifikationen

0*, 1, 2 oder 3 (Typenkennung) sowie
K*, V, B oder D (Ablagebereich) sowie
M (Markierung)

* bereits implizit vorgegeben

zu definieren. Nicht zulässige Spezifikationen werden mit einer Fehlermeldung angezeigt und überlesen.

	+3 412.25E3,
PI=	3.14159E,
	-0.5E+5/MV,
X=	.333E-4/V,
	-000.1E/DM,

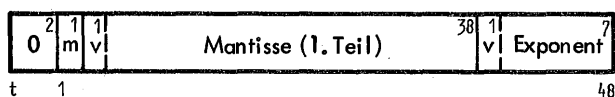
Bild 1.2 Beispiele für Gleitkommakonstanten einfacher Genauigkeit

1.2. Gleitkommakonstante doppelter Genauigkeit

Unter einer Gleitkommakonstanten doppelter Genauigkeit versteht man eine Gleitkommazahl, die ihrerseits aus einem Mantissenteil und einem Exponententeil besteht. Die Mantisse kann eine ganze Zahl, ein echter Bruch oder ein unechter Bruch sein. Der Exponent ist immer eine ganze Zahl. Gegenüber einer Gleitkommakonstanten einfacher Genauigkeit ist die Mantisse um ein TR 440-Ganzwort verlängert.

Gleitkommakonstanten doppelter Genauigkeit haben die Form

mDe m: Mantisse, ganze Zahl oder Bruch;
e: Exponent, ganze Zahl;
dezimal zur Basis 10;
(kann entfallen)



m = Marke
v = Vorzeichen

Bild 1.3 Interne Darstellung einer Gleitkommazahl doppelter Genauigkeit

Mit Gleitkommazahlen doppelter Genauigkeit wird ein Zahlenbereich von 10^{-155} bis 10^{+152} erfaßt. Die Genauigkeit beträgt 24 Dezimalstellen.

Ein Dezimalkomma ist stets als Punkt zu schreiben.

Intern wird eine Gleitkommakonstante doppelter Genauigkeit in z w e i aufeinanderfolgende TR 440-Ganzwörter übersetzt, und zwar werden in das erste Wort Mantisse und Exponent und in das zweite Wort die Verlängerung der Mantisse eingetragen. Das erste Wort hat Gleitkommastruktur mit der Typenkennung 0 und das zweite Wort Festkommastruktur mit der Typenkennung 1. Die Ablage erfolgt im Normalfall ohne Markierung im Konstantenbereich (K-Bereich).

Abweichungen vom Normalfall sind bei Gleitkommakonstanten doppelter Genauigkeit mit Hilfe der Spezifikation

K*, V, B oder D (Ablagebereich) sowie
M (Markierung)

* bereits implizit vorgegeben

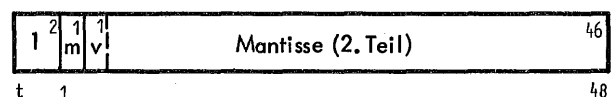
zu definieren. Nicht zulässige Spezifikationen werden mit einer Fehlermeldung angezeigt und überlesen.

Bei der Spezifikation M wird nur das erste Wort markiert. Das zweite ist stets unmarkiert.

PI=	3.14159265358979323846264D,
GD=	0.5D+8/MV,
	131D-15,
	.253D,

Bild 1.4 Beispiele für Gleitkommakonstanten doppelter Genauigkeit

In die Zahlen können Zwischenräume eingeschoben werden. Sie werden nicht mit übersetzt und können der übersichtlichen Gliederung dienen.



1.3. Festkommakonstante

Unter einer Festkommakonstanten versteht man eine Festkommazahl, die ihrerseits entweder eine ganze Zahl oder ein echter Bruch sein kann.

Bei echten Brüchen muß an Stelle des Dezimalkommata immer ein Punkt geschrieben werden. Dieser Punkt vor einer Zahl bedeutet stets, daß es sich um einen echten Bruch handelt, wobei es erlaubt ist, vor dem Dezimalpunkt eine Null zu schreiben. Auf das spätere Ergebnis hat diese Null keinen Einfluß.

ganze Zahl	echter Bruch
75,	.75,
+75,	+.75,
-75,	-.75,
	0.75,
	+0.75,
	-0.75,

Bild 1.5 Beispiele für die externe Schreibweise von Festkommakonstanten an Hand der Zahlen 75 bzw. 0.75

Intern wird eine Festkommakonstante in ein TR 440-Ganz- oder Halbwort mit Festkommastruktur und Typenkennung 1 umgewandelt. Die Ablage erfolgt ohne Markierung im Konstantenbereich K.

Die Typenkennung bei Halbwörtern unterliegt speziellen Regeln, die am Anfang des Abschnitts 1 näher erläutert sind.

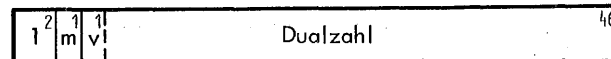
In die Zahlen können Zwischenräume eingeschoben werden. Sie werden nicht mit übersetzt und können der übersichtlichen Gliederung dienen.

1.3.1. Festkommakonstante mit Ganzwortlänge

Eine Festkommakonstante von der Länge eines Ganzwortes hat die Form

f f: Festkommazahl, dezimal;
ganze Zahl oder echter Bruch

Die Genauigkeit beträgt 13 Dezimalstellen; ganze Zahlen dürfen den Betrag von 70 368 744 177 633 nicht überschreiten.



t 1 46
m = Marke
v = Vorzeichen

Bild 1.6 Interne Darstellung einer Festkommakonstante ein-facher Länge

Festkommakonstanten ganzer Wortlänge können mit folgenden Spezifikationen versehen werden:

0, 1*, 2 oder 3 (Typenkennung) sowie
K*, V, B oder D (Ablagebereich) sowie
M (Markierung)

* bereits implizit vorgegeben

Nicht zulässige Spezifikationen werden mit einer Fehlermeldung angezeigt und überlesen.

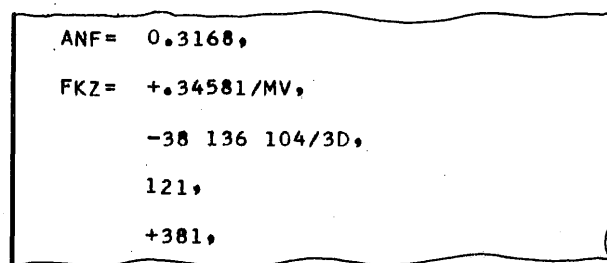


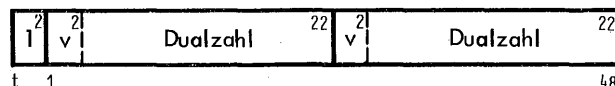
Bild 1.7 Beispiele für Festkommakonstanten

1.3.2. Festkommakonstante mit Halbwortlänge

Eine Festkommakonstante halber Wortlänge (Spezifikation H) hat die Form

f/H f: Festkommazahl, dezimal;
ganze Zahl oder echter Bruch

Die Genauigkeit beträgt 7 Dezimalstellen; ganze Zahlen dürfen den Betrag von 4 194 303 nicht überschreiten.



t 1 48
v = Vorzeichen

Bild 1.8 Interne Darstellung zwei Festkommazahlen halber Wortlänge

Festkommakonstanten halber Wortlänge können je nach Erfordernis mit folgenden Spezifikationen versehen werden:

- 0, 1*, 2 oder 3 (Typenkennung) sowie
- K*, V, B oder D (Ablagebereich) sowie
- H** (Halbwort) sowie
- G oder U (gerade/ungerade Adresse)

* bereits implizit vorgegeben
 ** zwingend erforderlich

```

  · AWERT=381461/VH,
  X=   +.321/G2VH,
       -0.165/H,
       0.361471368/H,
  
```

Bild 1.9 Beispiele für Festkommakonstanten halber Wortlänge

1.4. Tetradenkonstante

Eine Tetradenkonstante besteht aus einer Folge von Sedezimalzeichen, die in Apostrophe eingeschlossen werden, sie hat die Form

't'
 t: Sedezimalzeichen;
 max. 12 Zeichen bei Ganzwörtern
 max. 6 Zeichen bei Halbwörtern

Den Sedezimalzeichen werden intern Tetraden zugeordnet. Im Bild 1.10 ist die Zuordnung gezeigt.

Sedezimale	Tetrade	Sedezimale	Tetrade
0	0000	8	L000
1	000L	9	L00L
2	00L0	A	L0L0
3	00LL	B	L0LL
4	0L00	C	LL00
5	0L0L	D	LL0L
6	0LLO	E	LLLO
7	0LLL	F	LLLL

Bild 1.10 Binärdarstellung der Tetraden

Tetradenkonstante: '34A8F'

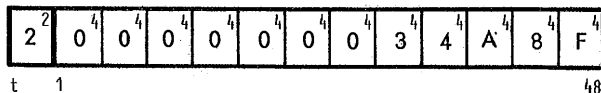


Bild 1.11 Beispiel für die interne Darstellung einer Tetradenkonstante

Im Normalfall werden Tetradenkonstanten in einem Ganzwort bzw. Halbwort rechtsbündig im K-Bereich abgelegt. Links wird ggf. mit Nullen aufgefüllt. Das Wort erhält die Typenkennung 2.

In die Folge der Sedezimalzeichen können beliebig Zwischenräume eingeschoben werden. Sie dienen nur dazu, eine übersichtliche Gliederung zu erhalten und werden nicht mit übersetzt.

1.4.1. Ganzwörter

Tetradenkonstanten mit Ganzwortlängen können bis zu 12 Sedezimalzeichen enthalten. Das Einsetzen in das Ganzwort erfolgt dabei rechtsbündig. Enthält die Tetradenkonstante weniger als 12 Zeichen, so wird das Wort links mit Nullen aufgefüllt.

Je nach Erfordernis können Tetradenkonstanten in Form von Ganzwörtern mit folgenden Spezifikationen versehen werden:

- 0, 1, 2* oder 3 (Typenkennung) sowie
- K*, V, B oder D (Ablagebereich) sowie
- L (linksbündig)

* bereits implizit vorgegeben

Nicht zulässige Spezifikationen werden mit einer Fehlermeldung angezeigt und überlesen.

Bei Verwendung der Spezifikation L (linksbündig) wird das Ganzwort ggf. r e c h t s mit Nullen aufgefüllt.

```

  MASKE= 'FFF 000 00F FFF',
  LZ=    'AF AF AF AF AF AF'/3V,
        '3F4B'/V3,
        '3F'/L1D,
  
```

Bild 1.12 Beispiele für Tetradenkonstante von Ganzwortlänge

1.4.2. Halbwörter

Tetradenkonstanten von Halbwortlänge können bis zu 6 Sedezimalzeichen enthalten. Sind weniger als 6 Zeichen angegeben, so wird der Rest des Halbwortes links mit Nullen aufgefüllt.

Die Tetradenkonstanten von Halbwortlänge können mit folgenden Spezifikationen versehen werden:

0, 1, 2* oder 3 (Typenkennung) sowie
 K*, V, B oder D (Ablagebereich) sowie
 L (linksbündig) sowie
 G oder U (gerade/ungerade Adresse) sowie
 H** (Halbwort)

* bereits implizit vorgegeben
 ** zwingend erforderlich

Bezüglich der Typenkennung bei Halbwörtern gilt das in Abschnitt 1. Gesagte.

X13= 'FF3B0'/H,
 '3'/HLG3D,
 ANF= 'B'/HL,
 '3F4'/HUL,

Bild 1.13 Beispiele für Tetradenkonstante von Halblänge

Oktadenkonstante: "TELEFUNKEN TR 440"

2	T	E	L	E	F	U
3	N	K	E	N	␣	T
3	R	␣	4	4	0	␣

␣ = Oktadenwert 0

Bild 1.14 Beispiel für die interne Darstellung einer Oktadenkonstante

Bei einer Oktadenkonstanten können folgende Spezifikationen angegeben werden:

0, 1, 2 oder 3* (Typenkennung) sowie
 K*, V, B oder D (Ablagebereich) sowie
 R (rechtsbündig)

* bereits implizit vorgegeben

Wird die Spezifikation R angegeben, so wird die Oktade so abgelegt, daß die letzte Oktade rechtsbündig im letzten Wort steht. Das erste Wort wird links mit dem Oktadenwert 0 (binär 0000 0000) aufgefüllt.

1.5. Oktadenkonstante

Bei einer Oktadenkonstante kann eine Folge von Zeichen angegeben werden, die beidseitig durch je zwei Apostrophe abgeschlossen wird. Jedem dieser Zeichen wird entsprechend dem Zentralcode eine Folge von 8 Bits (Oktade) zugeordnet. Die Oktadenkonstante hat die Form:

"k" k: Zeichen,
 Anzahl beliebig;

Es können nur die Zeichen angegeben werden, die auf dem zur Verfügung stehenden Eingabegerät darstellbar sind. Der Zwischenraum steht bei Oktadenkonstanten für sich selbst. Der Zeilenvorschub und Wagenrücklauf (bei Karten entspricht das dem Kartenende) sind bedeutungslos und werden nicht übersetzt.

Jedes Zeichen der Oktadenkonstante wird in eine Oktade übersetzt gemäß dem Zentralcode. Die Oktaden werden dabei linksbündig abgelegt, jeweils 6 Oktaden in einem Ganzwort. Bei mehr als 6 Oktaden wird mit dem nächsten Wort fortgefahren. Ist das letzte Wort nicht mit Oktaden gefüllt, so wird es mit dem Oktadenwert 0 (binär 0000 0000) aufgefüllt.



Soll eine Oktadenkonstante nur ein Halbwort belegen, so kann die Spezifikation H angegeben werden. Es sind dann folgende Spezifikationen erlaubt:

0, 1, 2 oder 3* (Typenkennung) sowie
 K*, V, B oder D (Ablagebereich) sowie
 R (rechtsbündig) sowie
 H** (Halbwort) sowie
 U, G (ungerade/gerade Adresse)

* bereits implizit vorgegeben
 ** zwingend erforderlich

In diesem Fall sind maximal nur 3 Zeichen erlaubt. Sind es weniger, so wird der Rest des Wortes mit dem Oktadenwert 0 (binär 0000 0000) aufgefüllt. Ist die Spezifikation R angegeben, so werden die Oktaden rechtsbündig abgelegt und links aufgefüllt.

```
BETRAG= 'DM 0.000,00',
PF= ' ,--' /R,
    '0' /HR2V,
    'SEITE NR.',
NR= '000',
```

Bild 1.16 Beispiele für Oktadenkonstante

1.6. Textkonstante

Bei der Oktadenkonstanten (Abschnitt 1.5.) können lediglich Zeichen angegeben werden, die auch auf dem jeweiligen Eingabegerät vorhanden sind. Mit Hilfe der Textkonstanten ist es möglich, jede beliebige Oktade einzugeben.

In der Textkonstanten kann jede Oktade dargestellt werden durch

- ihren Dezimalwert; er ist in der Zentralcodetabelle in jedem Feld in der rechten oberen Ecke angegeben und kann die Werte 0 bis 255 annehmen.
- ihren Sedezimalwert; er ergibt sich in der Zentralcodetabelle aus den beiden Koordinaten. Der Sedezimalwert ist in Apostrophe einzuschließen und darf die Werte von '00' bis 'FF' annehmen.
- ihren feststehenden Namen. Bei einer Reihe von Oktaden, die Steuerzeichen repräsentieren, ist dieser Name festgelegt. Er besteht aus zwei oder mehr Buchstaben und ist in der Zentralcodetabelle festgelegt.
- ihren vereinbarten Namen. Für jede Oktade kann im Quelltext mit Hilfe des Pseudobefehls TEXT ein Name vereinbart werden.

Die vorstehenden vier Möglichkeiten stehen jeweils für eine Oktade und sind ein Element in der Textkonstanten. Darüber hinaus kann auch angegeben werden

- eine Oktadenfolge; sie hat den gleichen Aufbau und die gleiche Bedeutung wie bei der Oktadenkonstanten. Es können alle Zeichen angegeben werden, die beim Eingabegerät vorhanden sind. Die Oktadenfolge ist auf beiden Seiten durch zwei Apostrophe begrenzt.

Die Oktadenfolge repräsentiert eine Folge von Oktaden und ist in der Textkonstanten ein Element.

Die Textkonstante hat die allgemeine Form

'(b)'
 b: Elemente (mehrere durch Komma getrennt),
 Dezimalwert der Oktade 0...255
 Sedezimalwert der Oktade '00'...'FF'
 Name der Oktade
 Oktadenfolge

Sie besteht aus einer Folge von Elementen der vorstehend beschriebenen Art, die durch die Zeichen '(' eingeleitet und durch die Zeichen ')' abgeschlossen wird. Die einzelnen Elemente sind durch Komma zu trennen.

In bezug auf Ablage und Spezifikationen gilt für die Textkonstante das gleiche wie für die Oktadenkonstante (siehe Abschnitt 1.5.).

LLOO OLLL	OOOL OOLO	Binärwert
'H'	-	Zeichen
-	VW	Name
199	18	Dezimalwert
'C7'	'12'	Sedezimalwert

Bild 1.17 Beispiel für die Darstellung von Oktaden in Textkonstanten

```
TEXT= '(ZW,'0',4,'A2','000,00',163) /F,
    '= A-WERT' /F,
    '(98','000,00)',
NAME= '(M',252,'EB','EB',228,241)',
```

Bild 1.18 Beispiele für Textkonstanten

Oktadennamen können mit Hilfe des Pseudobefehls TEXT für ein Quellenprogramm vereinbart werden. Der Pseudobefehl hat die Form

```
name = TEXT z      name: Name, den die Oktade erhalten soll
                   z: Dezimalwert der Oktade 0...255
```

Der Name muß - wie bei Namen üblich - mit einem Buchstaben beginnen. Es können den Buchstaben Ziffern folgen. Wird für eine Oktade, für die bereits ein Name vorhanden ist, ein zweiter Name definiert, so gelten beide. Wird ein Name, der bereits für eine Oktade vorhanden ist, für eine andere Oktade festgelegt, so gilt die jeweils letzte Festlegung.

```
'( ('775 K',238,237,242,243,224,237,
                               249) ',
```

```
'( ('775 K', 'EE', 'ED', 'F2', 'F3', 'E0',
                               'ED', 'F9') ',
```

```

:
O= TEXT 238,
N= TEXT 237,
S= TEXT 242,
T= TEXT 243,
A= TEXT 224,
Z= TEXT 249,
:
: ('775 K',O,N,S,T,A,N,Z)',
:
```

Alle Beispiele ergeben: 775 Konstanz

Bild 1.19 Beispiele für Textkonstanten und Pseudobefehl TEXT

1.7. Adressenkonstante

Mit Hilfe der Adressenkonstanten können Adressen in einem Halbwort abgelegt werden. Als Adressen sind alle zugelassen, die auch bei Befehlen zugelassen sind, die bei der Beschreibung im Adressenteil den Buchstaben n, m oder z haben. Es sind die im Abschnitt "Befehle und Adressierung", 2., 4. und 5. beschriebenen Adressen.

Darüber hinaus können nicht nur 16-Bit-Adressen, sondern alle 22-Bit-Adressen angegeben werden.

Die Adressenkonstante hat die Form

```
a/A,      a: Adresse einer Speicherzelle
```

Die Spezifikation A ist hier zwingend und kennzeichnet die Adresse als Adressenkonstante. Die Ablage erfolgt im Normalfall in einem Halbwort im K-Bereich mit der Typenkennung 2. Als Spezifikationen können angegeben werden:

```
0, 1, 2* oder 3 (Typenkennung) sowie
K*, V, B oder D (Ablagebereich) sowie
G oder U (gerade/ungerade Adresse) sowie
M (1. Bit setzen) sowie
N (2. Bit setzen)
```

* bereits implizit vorhanden

Die Adressen sind 22 Bits lang und werden in einem Wort mit 24 Bits rechtsbündig abgelegt. Die linken 2 Bits können durch die Spezifikationen M und N gesetzt werden. M setzt das linke Bit, das ist das 1. Bit und N setzt das 2. Bit.

Der Adressenkonstante kommt dann eine besondere Bedeutung zu, wenn mit 22-Bit-Adressen gearbeitet wird. Soll z.B. das Register B auf eine 22-Bit-Adresse gesetzt werden, so ist das mit dem Befehl

```
ADR = DBER/A
      TCB ADR,
```

möglich. Das gleiche kann auch als Literal geschrieben werden.

```
TCB (DBER/A),
```

In beiden Fällen wird die Adresse DBER in einem Halbwort abgelegt und mit dem Befehl TCB ins Register B gebracht.

Ein weiteres Anwendungsgebiet liegt bei den Befehlen SE, SFBE und SUE, bei denen das Sprungziel nicht im Adressenteil steht, sondern in einem Halbwort. Die Zieladresse wird mit einer Adressenkonstante abgespeichert.

```

ANTON+20/A,

ADR= ANFANG+X1/A,

TCB (EMIL/A),

B2 (VSB/A),

80000/A,

DIV= ANF-END/A,
      '3F 4B 3C'+8/A,
```

Bild 1.20 Beispiele für Adressenkonstanten

1.8. Bitfeldkonstante

Mit Hilfe der Bitfeldkonstanten können in Ganz- oder Halbwörtern beliebige Bitfelder besetzt werden.

Sie hat die Form

(i/e), i/e: Bitfeldangabe; kann mehrmals angegeben werden, durch Komma trennen
 i: Inhalt des Bitfeldes
 e: Länge des Bitfeldes

Innerhalb der Klammer kann die Bitfeldangabe mehrfach gemacht werden. Die Angaben sind durch Komma voneinander zu trennen. Die Bitfeldangabe besteht aus Bitfeldinhalt und Bitfeldlänge, die durch Schrägstrich getrennt sind.

Eine Bitfeldkonstante besteht aus einer Liste von einzelnen Bitfeldangaben. Eine einzelne Bitfeldkonstante darf maximal ein Ganzwort lang sein. Die Summe aller Bitfeldlängen darf also nicht größer als 48 sein.

Der Beginn eines Bitfeldes ist identisch mit dem auf das Ende des davorliegenden Bitfeldes folgenden Bit. Das nächste Bitfeld schließt also immer unmittelbar an das vorhergehende an. Der Bitfeldinhalt wird im Bitfeld rechtsbündig abgelegt. (Überzählige Bits sind nicht gesetzt.)

Der Inhalt des Bitfeldes kann angegeben werden

- Dezimal
- Sedezimal
- Oktadenfolge
- Textfolge
- Differenzadresse
- Indexadresse
- Summenadresse aus den vorstehenden Elementen

Näheres zu diesen Angaben ist im Abschnitt "Befehle und Adressierung", 4. zu entnehmen.

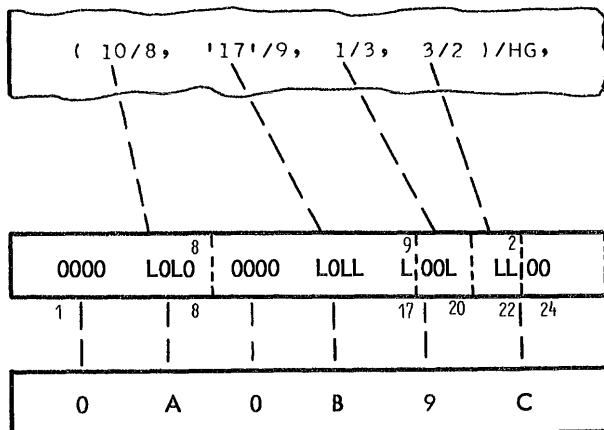


Bild 1.21 Bitfeldangabe mit interner Darstellung

Im Normalfall wird die Bitfeldangabe in einem Ganzwort mit der Typenkennung 2 im K-Bereich abgelegt. Die folgenden Spezifikationen sind erlaubt:

0, 1, 2* oder 3 (Typenkennung) sowie
 K*, V, B oder D (Ablagebereich) sowie
 H (Halbwort) sowie
 G** oder U** (gerade/ungerade Adresse) sowie
 F (Fortsetzung)

* bereits implizit vorgegeben
 ** nur wenn gleichzeitig H

Eine Bitfeldkonstante kann mit F spezifiziert werden. Ist in derselben Adressenzone, in der die Bitfeldkonstante abgelegt werden soll, vorher als letzte Einheit eine mit F spezifizierte Konstante (auch Bitfeldkonstante) abgelegt worden, so schließt die Bitfeldkonstante unmittelbar an diese an. Somit können Bitfelder auch über Ganzwortgrenzen hinweg abgelegt werden.

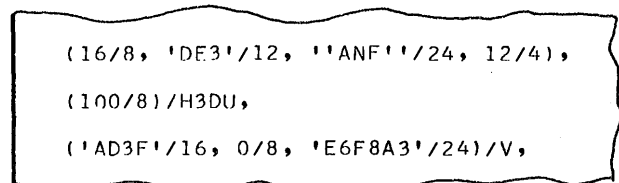


Bild 1.22 Beispiele für Bitfeldangabe

1.9. Konstante im Befehlsformat

Die Darstellung eines Befehls im TAS-Interncode (siehe Abschnitt "Befehle und Adressierung", 1.2.) erlaubt es, Halbwörter in bestimmten Bitfeldern zu benutzen, wenn die Bitfeldteilung mit der Teilung einer der TR 440-Befehle übereinstimmt.

Der TAS-Interncode hat die Form

I 't' (c) adr t: max. 2 Sedezimale
 c: Befehlscode
 adr: Adressenteil

Der Wert t wird in die linken 8 Bits abgelegt. Der mit c angegebene Befehlscode wird nicht übersetzt, sondern dient dazu, die Art des Adressenteils zu bestimmen. Der Adressenteil wird gemäß dem Befehlscode c übersetzt und in die rechten 16 Bits eingesetzt.

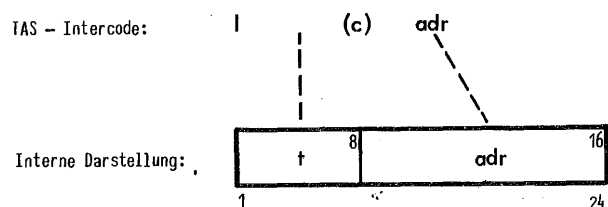


Bild 1.23 Prinzip des TAS-Interncodes

Im Normalfall wird der TAS-Interncode in einem Halbwort mit Typenkennung 2 im B-Bereich abgelegt. Folgende Spezifikationen sind erlaubt:

- 0, 1, 2* oder 3 (Typenkennung) sowie
- K, V, B* oder D (Ablagebereich) sowie
- G oder U (gerade/ungerade Adresse)

* bereits implizit vorgegeben

In dem hier beschriebenen Fall soll der TAS-Interncode als Konstante gelten und daher nicht im B-Bereich abgelegt werden. Es ist also eine der Spezifikationen K, V oder D anzugeben.

Im Bild 1.24 sind drei Beispiele angegeben. Bei dem ersten wird eine Halbwortadresse auf Grund der Angabe (B2) in die rechten 16 Bits gebracht. In die linken 8 Bits wird der Wert '10' gebracht. Beim zweiten Beispiel können auf Grund der Angabe (SL) die rechten 8 Bits einzeln gesetzt werden, indem die Nummern 1 bis 8 angegeben werden. Die mittleren 8 Bits werden auf den Wert -100 gesetzt und die linken auf den Wert '34'. Im dritten Beispiel kann jedes der rechten 16 Bits einzeln gesetzt werden.

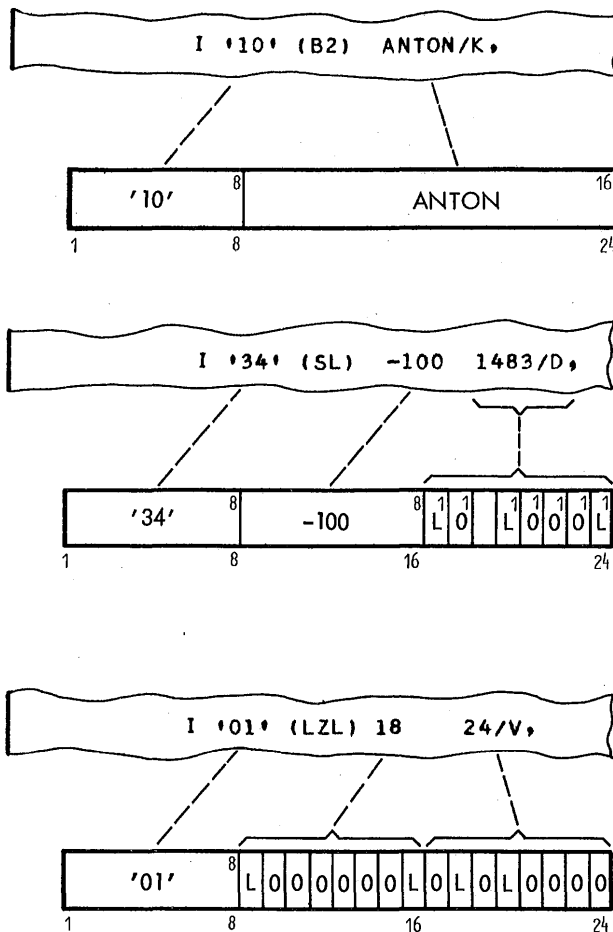


Bild 1.24 Beispiele für Konstanten im Befehlsformat

2. AUFBAU VON QUELLENPROGRAMMEN

Die einzelnen Informationseinheiten eines TAS-Quellenprogramms können aus Befehlen, Konstanten und Pseudobefehlen bestehen. In den folgenden Abschnitten wird beschrieben, wie mit Hilfe von Pseudobefehlen der Übersetzungsvorgang von Quellenprogrammen gesteuert werden kann.

2.1. Programmanfang

Ist der erste Pseudobefehl SEGM benannt, so erhält das Quellenprogramm diesen Namen. Besitzt dieser Pseudobefehl mehrere Namen, dann wird sein erster Name der Quellenprogrammname. Wenn der erste Pseudobefehl SEGM nicht benannt ist, erhält das Quellenprogramm keinen Namen. In der Kopfzeile des Übersetzungsprotokolls bleibt die Position "Programmname" unbesetzt.

Der Name des Quellenprogramms wird der Name des erzeugten Montageobjekts, falls im UEBERSETZE-Kommando kein anderer Montageobjektname explizit vereinbart wird (siehe Kommandohandbuch). Bei einem namenlosen Quellenprogramm wird der Standardname STDHP verwendet, wenn im UEBERSETZE-Kommando statt eines Namens die Spezifikation -STD- angegeben wird.

2.2. Programmende

Das Ende eines Quellenprogramms wird dem Assembler durch den Pseudobefehl

ENDE

angezeigt. Der Übersetzungsvorgang wird beendet. Damit ist er der letzte Befehl des Quellenprogramms. Fehlt dieser Befehl, so wird vom Assembler eine Fehlermeldung gegeben und ein Pseudobefehl ENDE eingefügt.

Beim Erreichen des Pseudobefehls ENDE müssen alle mit F spezifizierten Konstanten abgeschlossen sein. Offene F-Konstanten werden abgeschlossen und eine Fehlermeldung wird ausgegeben.

2.3. Segmente

Die Quellenprogramme können in Segmente gegliedert werden. Dadurch besteht die Möglichkeit, den Geltungsbereich von Namen, die für symbolische Adressen stehen, auf einen Teil des Quellenprogramms zu begrenzen. Jedes Segment beginnt mit dem Pseudobefehl SEGM.

name = SEGM name: Name des Segments

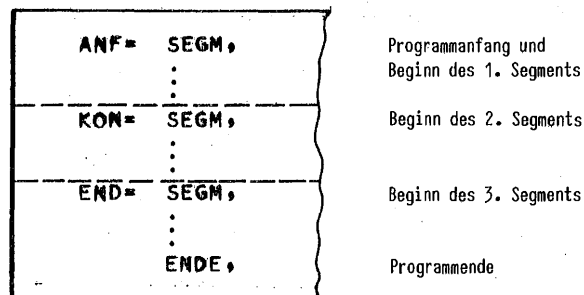


Bild 2.1 Beispiel für eine Programmaufteilung in Segmente

Die Segmente können benannt werden. Ein neuer Pseudobefehl SEGM schließt das vorhergehende Segment ab. Lokale Namen gelten nur in dem Segment, in dem sie festgelegt wurden. Im neuen Segment gelten alle bisher festgelegten lokalen Namen nicht mehr. Dagegen sind globale Namen im gesamten Programm ansprechbar. Kommt in einem Segment ein globaler Name auch als lokaler Name vor, dann ist der Geltungsbereich des globalen Namens in diesem Segment aufgehoben.

Der Name eines Segments ist auch gleichzeitig eine symbolische Adresse für das nächste freie Halbwort in derjenigen Adressenzone, in die ein unmittelbar hinter dem Pseudobefehl SEGM stehender, nicht spezifizierter Befehl abgelegt werden würde.

Ein besonderer Fall ist zu beachten. Folgt auf einen benannten Pseudobefehl SEGM unmittelbar ein Pseudobefehl STARR, so wird im allgemeinen der Name des Segments nicht einer Informationseinheit derselben Zone zugeordnet, der die folgenden Informationseinheiten zugeordnet werden. Dadurch kann man die folgenden Informationseinheiten nicht mit dem Namen des Pseudobefehls SEGM aufrufen. Dies kann dadurch umgangen werden, daß man den Pseudobefehl STARR vor den benannten Pseudobefehl SEGM legt.

2.4. Voreinstellungen für Operatoren

Die Voreinstellungen dürfen nur einmal für den Operator angegeben werden. Eine Ausnahme bildet der Pseudobefehl STRUKT, der in jedem Quellenprogramm einmal vorkommen darf.

Für den Start eines Operators müssen die sechs folgenden Angaben gemacht werden:

- die Adresse, auf die das Indexbasisregister X gesetzt werden soll,
- die Indexadresse, auf die das Unterprogrammregister U gesetzt werden soll,

- die Fehleradresse, auf die beim Auftreten eines Fehlers gesprochen werden soll,
- die Vorbesetzung der vereinbarten leeren Speicherzellen,
- die Startadresse, bei der der Operatorlauf beginnen soll,
- die Adresse, bei der der Kontrollblock für Testversionen liegt.

Das Indexbasisregister wird mit dem Pseudobefehl XBASIS gesetzt.

XBASIS m m: Adresse des Halbwortes, bei dem der Indexbereich beginnt

Die Adresse gibt den Beginn des Indexbereichs an. Der Indexbereich kann an einer beliebigen Stelle in einem nicht schreibgeschützten Ablagebereich liegen. Für den Indexbereich müssen mit dem Pseudobefehl ASP oder DSP soviele Halbörter freigehalten werden, wie Indexzellen gebraucht werden.

Das Unterprogrammregister wird mit dem Pseudobefehl UNTPR gesetzt.

UNTPR i i: Indexadresse, auf die das Unterprogrammregister gesetzt wird

Die Fehleradresse wird mit dem Pseudobefehl ALARM angegeben.

ALARM m m: Adresse eines Befehls, bei dem ein Fehlerprogramm beginnt

Tritt während des Operatorlaufs ein Fehler auf, so startet der Abwickler den Operator auf seiner Alarmadresse unter gleichzeitiger Übergabe von Fehlerinformationen.

Alle vereinbarten leeren Speicherzellen werden mit dem Pseudobefehl VORBES auf bestimmte Werte vorbesetzt. Dies gilt für die mit den Pseudobefehlen ASP und DSP vereinbarten Speicherbereiche sowie für alle anderen Adresszonen und für Gebiete.

VORBES (t, 's') t: Typenkennung
 s: 12 Sedezimalzeichen

Die allgemeine Vorbesetzung kann aber auf explizit definierten Gebieten abgeändert werden (siehe Abschnitt 5).

Die Adresse, bei der der Operatorlauf beim Start beginnen soll, wird mit dem Pseudobefehl START angegeben.

START adr adr: Startadresse, bei der der Operatorlauf beginnen soll

Die im Pseudobefehl STRUKT mitgegebene Information wird nur in Testversionen im Alarmfall bei der Rückverfolgung der Unterprogrammaufrufhierarchie ausgewertet.

STRUKT (typ, name) typ: Zahl 1...15 oder eine Tetrade
 ≠ 0
 name: Kontrollblockname

Der Typ gibt an, nach welchen Unterprogrammkonventionen das Quellenprogramm aufgebaut ist. Typ 1 bedeutet, daß das Quellenprogramm ein Unterprogramm nach den allgemeinen Unterprogrammkonventionen ist und daß der Kontrollblock bei der angegebenen Adresse liegt (siehe Abschnitt D, Programmteile). Der Parameter Typ=1 darf fehlen. Der Assembler gibt dann aber eine Fehlermeldung aus. Ist der Typ≠1, so darf man auf den Kontrollblocknamen verzichten.

```

VOR= SEGM,
      XBASIS  XB,
XB=  ASP    30,
      UNTPR   5,
      ALARM   AADR,
      VORBES  (1, '000000000000'),
      START   ANF,
      STRUKT  (1, KONTROLLBL),
      ...

```

Bild 2.2 Beispiel für die Voreinstellung

Man kann sich die Arbeit sparen, die Pseudobefehle für die Voreinstellung zu schreiben, wenn man das Makro R&RAHMEN benutzt. Zweckmäßigerweise ruft man das Makro R&RAHMEN implizit mit der Spezifikation SPRACHE = TASR im UEBERSETZE-Kommando auf. Die Wirkung des Aufrufs ist:

- Ein Speicherbereich für 256 Indexzellen wird reserviert und ein Pseudobefehl XBASIS mit der Anfangsadresse dieses Bereichs erzeugt.
- Ein Pseudobefehl UNTPR 247 wird erzeugt.
- Ein Pseudobefehl VORBES(3, 'FFFFFFFF') wird erzeugt.
- Ein Pseudobefehl ALARM zur Behandlung von Alarmen und SSR-Fehlern wird erzeugt, die Montageobjekte S&CC und S&SRF angeschlossen und die erforderlichen Anfangsaufrufe durchgeführt.
- Ein Pseudobefehl START wird erzeugt.

Wenn das Makro R&RAHMEN benutzt wird, dürfen die Pseudobefehle XBASIS, UNTPR, VORBES, ALARM und START nicht mehr geschrieben werden.

Näheres zum Makro R&RAHMEN und den Wirkungen ist dem Abschnitt D 10, Standardrahmen für TAS-Programme zu entnehmen.

3. VEREINBARUNG VON SPEICHERBEREICHEN

Im Adressenraum können Teile freigehalten werden, in denen während des Programmlaufs Informationen gespeichert werden können.

3.1. Arbeitsspeicher

Arbeitsspeicher können mit dem Pseudobefehl ASP freigehalten werden. Dabei können entweder die freizuhaltenden Halbwörter oder die Teilseiten angegeben werden.

ASP h h: Anzahl der freizuhaltenden Halbwörter
 s: Spezifikation
ASP h/s s₁: Spezifikation der Anfangsadresse
 leer = nächste freie Adresse
 G = nächste gerade Adresse
 U = nächste ungerade Adresse
 s₂: Ablagespezifikation
 leer = Variablenbereich implizit zugeordnet
 K = Konstantenbereich
 V = Variablenbereich
 B = Befehlsbereich
 D = Datenbereich

In diesem Fall wird ein Arbeitsspeicher von h Halbwörtern freigehalten. Die Arbeitsspeicher werden normalerweise im Variablenbereich angeordnet und sind dann mit 16-Bit-Adressen adressierbar. Mit der Spezifikation s₁ kann angegeben werden, ob der Arbeitsspeicher mit einer geraden oder ungeraden Adresse beginnen soll. Abweichend vom Normalfall können mit Hilfe der Ablagespezifikation s₂ die Arbeitsspeicher auch in einem anderen Teil des Adressenraums angelegt werden.

Eine Benennung des Arbeitsspeichers ist zulässig und zweckmäßig. Die Anfangsadresse des Arbeitsspeichers kann dann unter dem Namen aufgerufen werden.

Die Größe eines Arbeitsspeicherbereichs kann auch in Teilseiten angegeben werden. In diesem Fall wird der Arbeitsspeicher so im Adressenraum angeordnet, daß sein Beginn entsprechend dem Typ der Teilseitenangabe mit einem Ganz-, Halb-, Viertel- oder Achtelseitenbeginn zusammenfällt.

ASP pKa p: Anzahl der freizuhaltenden Teilseiten
 a: Angabe der Teilseite
ASP pKa/s leer = Ganzseite 2048 Halbwörter
 2 = Halbseite 1024 Halbwörter
 4 = Viertelseite 512 Halbwörter
 8 = Achtelseite 256 Halbwörter
 s: Ablagespezifikation
 leer = Variablenbereich implizit zugeordnet
 K = Konstantenbereich
 V = Variablenbereich
 B = Befehlsbereich
 D = Datenbereich

Durch die Ablagespezifikation s kann der Arbeitsspeicher abweichend vom Normalfall auch in einem anderen Teil des Adressenraums außerhalb des Variablenbereichs angelegt werden.

```
TSP= ASP 40,
ARB= ASP 17/U,
ZWSP= ASP 200/GB,
TRP= ASP 3K,
SUSP= ASP 7KB/B,
```

Bild 3.1 Beispiele für Vereinbarung von Arbeitsspeichern

3.2. Datenspeicher

Datenspeicher können mit dem Pseudobefehl DSP freigehalten werden. Es gelten sinngemäß die Konventionen des Pseudobefehls ASP. Datenspeicher werden aber im Gegensatz zu Arbeitsspeichern normalerweise im Datenbereich freigehalten und sind dann nur indirekt mit einer 22-Bit-Adresse adressierbar.

```
DAT= DSP 700,
LIST= DSP 300/G,
DAP= DSP 9K2,
RSP= DSP 5K4/B,
```

Bild 3.2 Beispiele für Vereinbarung von Datenspeichern

4. VEREINBARUNG VON ADRESSENZONEN UND GEBIETEN

Adressenzonen sind vom Assembler eingeführte Adressenräume. Der Assembler richtet eine Adressenzone implizit erst bei Bedarf ein. Explizit können Adressenzonen durch die Pseudobefehle ZONE, CZONE oder FZONE vereinbart werden.

Implizit kann der Assembler maximal 13 verschiedene Adressenzonen einrichten. Diese Adressenzonen haben keinen Namen. Zu Beschreibungszwecken werden sie bezeichnet:

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

Dabei bedeutet jeweils der erste Buchstabe eine Adressierungsbedingung:

K: schreibgeschützt, mit 16-Bit-Adressen adressierbar, (Konstanten),

V: nicht schreibgeschützt, mit 16-Bit-Adressen adressierbar, (Variable),

B: schreibgeschützt, alle diese Adressenzonen kommen in eine Großseite, (Befehle),

D: nicht schreibgeschützt, mit 22-Bit-Adressen adressierbar, (Daten).

Der zweite Buchstabe bezeichnet den impliziten Literal-schlüssel:

- 0 Informationseinheiten außerhalb von Literalen
- 1 Informationseinheiten in Literalen (Literale 1. Ordnung)
- 2 Informationseinheiten in Literalen, die wiederum in Literalen stehen (Literale 2. Ordnung)
- G Ganzwortlitterale, Literale, die ein Ganzwort einnehmen und im K-Bereich liegen, werden bei mehrfachem Auftreten nur einmal abgelegt (nur im K-Bereich möglich)

Darüber hinaus gibt es noch Adressierungsbedingungen, die nur für explizit definierte Adressenzonen Bedeutung haben:

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

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

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

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

I: nicht schreibgeschützt, mit 16-Bit-Adressen adressierbar, (Indexzone).

Der Anfang der Indexzone wird mindestens 256 Adressenstellen vor dem Ende des Gebiets abgelegt, in dem die Indexzone angeordnet wird.

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

Das Montageprogramm verbindet die zu einem Operator gehörigen Adressenzonen zu einem Operatorkörper. Dieser ist in Gebiete gegliedert. In jedem Gebiet werden bestimmte Adressenzonen zusammengefaßt.

Für die implizit vereinbarten Adressenzonen und die explizit vereinbarten Adressenzonen, über deren Anordnung nicht explizit verfügt wird, erstellt das Montageprogramm implizit Gebiete. Explizit können Gebiete durch den Pseudobefehl GEBIET vereinbart werden.

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

Die Anzahl der explizit und implizit definierten Adressenzonen ist pro Montageobjekt auf 31 begrenzt.

4.1. Adressenzonen

Eine Adressenzone kann explizit mit dem Pseudobefehl ZONE vereinbart werden.

name= ZONE a

name: Name der Adressenzone

a: Adressierungsbedingung

K, B, DK, F

V, BV, D, FV oder I

4.2. Commonzone

Mit dem Pseudobefehl CZONE kann eine Commonzone definiert werden.

```

name = CZONE a      name: Name der Commonzone
                    a: Adressierungsbedingung
                      K, B, DK, F
                      V, BV, D, FV oder I

```

Bei der Montage von Quellenprogrammen zu einem Operator werden alle Commonzonen, die den gleichen Namen tragen, als identisch angesehen und daher nur einmal im Adressenraum angeordnet.

Der Umfang des zugewiesenen Adressenraums wird nach der längsten aller gleichnamigen Commonzonen bemessen.

4.3. Freihaltezone

Eine Freihaltezone kann mit dem Pseudobefehl FZONE angelegt werden.

```

name = FZONE a      name: Name der Freihaltezone
                    a: Adressierungsbedingung
                      K, B, DK, F
                      V, BV, D, FV oder I

```

Die Freihaltezone wird zu Beginn eines Operatorlaufs noch nicht belegt. Sie beansprucht daher zwar einen Teil des Adressenraums aber noch keinen physikalischen Speicherraum. Sie wird für eine spätere Verwendung freigehalten. Die Informationseinheiten, die einer Freihaltezone durch einen Pseudobefehl ABLAGE zugeordnet sind, werden daher vom Assembler nicht in die interne Form übersetzt. Sie dienen nur zur Festlegung der Länge der Freihaltezone und eventuell zur Vereinbarung von Namen für Adressen der Freihaltezone.

```

ADX=  ZONE  D,
C3=   CZONE V,
F24=  FZONE K.

```

Bild 4.1 Beispiel für die explizite Vereinbarung von Adressenzonen

4.4. Gebiet

Durch den Pseudobefehl GEBIET wird ein Gebiet explizit vereinbart.

```

GEBIET (p)  p: Gebietsparameter, mehrere Angaben
              durch Komma trennen

```

Es können mehrere Gebietsparameter angegeben werden. Die Angabe der Gebietsparameter ist nicht zwingend. Mehrere Parameter können in beliebiger Reihenfolge geschrieben werden, wenn sie mit ihrem Namen angegeben werden. Wird aber der Name mit dem Gleichheitszeichen fortgelassen, dann wird die nachfolgende Reihenfolge der Parameter vorausgesetzt. Es ist dadurch eine mehrfache Besetzung eines Parameters möglich. In einem solchen Fall ist der letzte Parameterwert gültig. Es wird dann eine Warnung ausgegeben. Folgende Gebietsparameter sind zulässig:

```

p0: operatorspezifischer Gebietsname
      ONAME = name  name: operatorspezifischer
                    Gebietsname

```

Der operatorspezifische Gebietsname muß eindeutig sein in allen zum Operator gehörenden Montageobjekten. Es wirken nur die ersten 6 Zeichen unterscheidend.

```

p1: prozeßspezifischer Gebietsname
      PNAME = name  name: prozeßspezifischer
                    Gebietsname

```

Zusätzlich zum operatorspezifischen Gebietsnamen oder stattdessen kann auch ein prozeßspezifischer Gebietsname angegeben werden. Dieser muß in dem Prozeß, zu dem der Operator gehört, eindeutig sein. Mindestens einer der Parameter p₀ und p₁ muß angegeben werden. Es wirken nur die ersten 6 Zeichen unterscheidend.

```

p2: Lebensdauer des Gebietes
      LD = d      d: Lebensdauer des Gebietes
                  L = Laufzeitgebiet
                  D = Dauergebiet
                  Parameter ist mit D vorbesetzt

```

```

p3: Adressierungsbedingung
      ADR = a      a: Adressierungsbedingung
                  K, B, DK, F
                  V, BV, D oder FV
                  Parameter ist mit K vorbesetzt

```

Alle Gebiete mit ADR = B oder ADR = BV werden derselben Großseite zugeordnet wie die B-Adressenzonen.

```

p4: gewünschte Verarbeitungsklasse
      VK1 = v

```

```

p5: geforderte Verarbeitungsklasse
      VK2 = v      v: Verarbeitungsklasse
                  KS = Kernspeicher
                  MK = Massenkernspeicher,
                      nicht direkt adressierbar
                  IR = Trommel
                  PL = Platte
                  Beide Parameter sind mit KS vorbesetzt

```

p₆ : Lagerklasse

LK = k k: Lagerklasse
MK = Massenkernspeicher,
nicht direkt adressierbar
TR = Trommel
PL = Platte
HG = Hintergrundspeicher
Parameter ist mit PL vorbesetzt

p₇ : Vorbesetzung

VB = (t, 's') t: Typenkennung
0, 1, 2, 3
s: 12 Sedezimalzeichen

Mit diesem Parameter kann die Vorbesetzung jedes Ganzwortes im Gebiet angegeben werden. Diese Vorbesetzung überschreibt für dieses Gebiet die allgemeine Vorbesetzung, die mit dem Pseudobefehl VORBES eingestellt wird.

p₈ : Paßwort für Lesezugriff

PWL = b b: Bedingung
name = Paßwort für Lesezugriff
0 = nicht als Fremdgebiet beschaffbar
leer = als Fremdgebiet ohne Paßwort beschaffbar

Mit diesem Parameter kann die Bedingung für die Beschaffung des Gebietes als Fremdgebiet zum Lesen bestimmt werden. Ist der Parameter mit einem Paßwort angegeben, so wird das Gebiet nur bei Anlieferung des Paßwortes als Fremdgebiet zum Lesen freigegeben. In diesem Fall muß auch der Parameter PWLS angegeben werden. Steht statt des Paßwortes die Ziffer 0, so ist das Gebiet nicht als Fremdgebiet zum Lesen beschaffbar. Es muß dann auch PWLS = 0 sein. Fehlt der Parameter, so kann sich jeder Prozeß das Gebiet zum Lesen beschaffen.

p₉ : Paßwort für Lese- und Schreibzugriff

PWLS = b b: Bedingung
name = Paßwort für Lese- und Schreibzugriff
0 = nicht als Fremdgebiet beschaffbar
leer = als Fremdgebiet ohne Paßwort beschaffbar

Mit diesem Parameter kann die Bedingung für die Beschaffung des Gebiets als Fremdgebiet zum Lesen und Schreiben bestimmt werden. Ist der Parameter mit einem Paßwort angegeben, so wird das Gebiet nur bei Anlieferung des Paßwortes als Fremdgebiet zum Lesen und Schreiben freigegeben. Steht statt des Paßwortes die Ziffer 0, so ist das Gebiet nicht als Fremdgebiet zum Lesen und Schreiben beschaffbar. Fehlt der Parameter, so kann sich jeder Prozeß das Gebiet zum Lesen und Schreiben beschaffen.

```
GEBIET ( ONAME=OP1, PNAME=PRO6, LD=L, ADR=D, VK1=TR, VK2=PL, LK=HG,
        VB=( 3, '000000000000' ), PWL=LESE, PWLS=LESCH ),
GEBIET ( OP1, PRO6, L, D, TR, PL, HG, ( 3, '000000000000' ), LESE, LESCH ),
```

Bild 4.2 Beispiel für die explizite Vereinbarung von Gebieten

Durch den Pseudobefehl GEBIET explizit vereinbarte Gebiete müssen auch explizit im Adressenraum angeordnet werden (durch den Pseudobefehl GEBAN, siehe B 3.4.4). Ein explizit vereinbartes Gebiet ist nur dann sinnvoll, wenn ihm auch Zonen zugeordnet werden (durch den Pseudobefehl ZONAN, siehe B 3.4.3).

4.5. Lücke

Mit dem Pseudobefehl LUECKE wird eine Lücke im Adressenraum vereinbart.

LUECKE name a name = Lückename, max. 6 Zeichen
a = Adressierungsbedingung
K, B, DK, F
V, BV, D oder FV
mit D vorbesetzt

Eine Lücke beginnt mit einer Seite und umfaßt ein Vielfaches einer Seite. Der Lücke wird zu Beginn des Operatorlaufs kein physikalischer Speicher zugewiesen. Sie verhält sich zum Gebiet wie die Freihaltezone zu den anderen Zonen. In der Lücke können nur Freihaltezonen angeordnet werden. Alle Freihaltezonen werden in Lücken angeordnet. Die Größe der Lücke ergibt sich aus den Freihaltezonen, die ihr zugeordnet sind.

```
LUECKE LA12 V,
LUECKE FL D,
```

Bild 4.3 Beispiel für die explizite Vereinbarung von Lücken

Durch den Pseudobefehl LUECKE vereinbarte Lücken müssen explizit im Adressenraum angeordnet werden (durch den Pseudobefehl GEBAN, siehe B 3.4.4). Eine Lücke ist nur dann sinnvoll, wenn ihr auch Freihaltezonen zugeordnet werden (durch den Pseudobefehl ZONAN, siehe B 3.4.3).

5. STEUERUNG DER ABLAGE

Es bestehen folgende drei Möglichkeiten, die Ablage eines Befehls, einer Konstanten oder die durch die Pseudobefehle ASP und DSP freigehaltenen Speicherbereiche zu steuern:

- Die implizite Ablagespezifikation:
Befehle in den B-Bereich
Konstanten in den K-Bereich
Speicherplatz durch Pseudobefehl ASP im V-Bereich
Speicherplatz durch Pseudobefehl DSP im D-Bereich
- Die Ablagespezifikation durch den Pseudobefehl STARR. Sie überschreibt die implizite Ablagespezifikation (siehe Abschnitt 5.1.).
- Die explizite Ablagespezifikation der Informationseinheit. Sie überschreibt die implizite Zuordnung und auch die durch den Pseudobefehl STARR gegebene Zuordnung.

Implizit besitzt jede Informationseinheit mit Ausnahme der meisten Pseudobefehle einen Ablageschlüssel und einen Literalschlüssel.

Impliziter Ablageschlüssel:

- K Konstanten, schreibgeschützt, mit 16-Bit-Adressen adressierbar
- V Variable, nicht schreibgeschützt, mit 16-Bit-Adressen adressierbar
- B Befehle, schreibgeschützt, alle Adressenzonen in einer Großseite
- D Daten, nicht schreibgeschützt, mit 22-Bit-Adressen adressierbar

Impliziter Literalschlüssel:

- 0 Informationseinheiten außerhalb von Literalen
- 1 Informationseinheiten in Literalen (Literals 1. Ordnung)
- 2 Informationseinheiten in Literalen, die wiederum in Literalen stehen (Literals 2. Ordnung)
- G Ganzwortliterals, Literale, die ein Ganzwort einnehmen und im K-Bereich liegen, werden bei mehrfachem Auftreten nur einmal abgelegt (nur im K-Bereich möglich)

Ablageschlüssel und Literalschlüssel bilden zusammen die Ablagekennung:

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

Der implizite Ablageschlüssel kann explizit verändert werden. Damit ändert sich auch die Ablagekennung. Der Literalschlüssel ist im allgemeinen unveränderlich.

Nur bei der Ablagekennung KG wird der Literalschlüssel sinngemäß in 0, 1 oder 2 abgeändert, wenn der Ablageschlüssel explizit verändert wird. (Zur Steuerung der Ablage siehe auch Abschnitt Aufbau, 3. Gliederung des Adressenraums.)

5.1. Verteilung der Informationseinheiten auf die Adressenzonen

Informationseinheiten, die in Literalen vorkommen, werden von den anderen Informationseinheiten getrennt. Zu diesem Zweck gibt es verschiedene Adressenzonen. Die Zuordnung einer Informationseinheit zu einer bestimmten Adressenzone erfolgt aufgrund ihrer Ablagekennung. Jede Informationseinheit besitzt implizit eine Ablagekennung. Diese kann explizit verändert werden durch eine Ablagespezifikation oder durch den Pseudobefehl STARR.

Während die explizite Angabe einer Ablagespezifikation nur immer für eine Informationseinheit gilt, kann mit Hilfe des Pseudobefehls STARR einer ganzen Gruppe von Informationseinheiten eine bestimmte Ablagespezifikation zugeordnet werden.

STARR s s: Ablagespezifikation
 K, V, B oder D

Allen auf den Pseudobefehl STARR folgenden Informationseinheiten wird die angegebene Ablagespezifikation zugeordnet. Eine Ausnahme bilden die Informationseinheiten, die eine eigene explizite andere Ablagespezifikation führen. Die Wirkung des Pseudobefehls STARR wird aufgehoben durch einen neuen Pseudobefehl STARR oder durch den Pseudobefehl

STEND

Er ist eine Anweisung an den Assembler, die durch einen vorhergehenden Pseudobefehl STARR gegebene Anweisung aufzuheben.

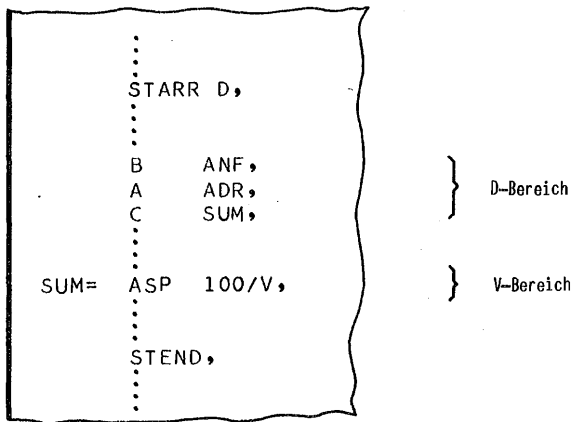


Bild 5.1 Beispiel für die Ablage von Informationseinheiten

Die Adresszonen werden bei Bedarf implizit vom Assembler vereinbart. Daneben gibt es die Möglichkeit, mit Hilfe der Pseudobefehle ZONE, CZONE und FZONE weitere Adresszonen explizit festzulegen (siehe Abschnitt 4). Mit dem Pseudobefehl ABLAGE lassen sich Informationseinheiten aufgrund ihrer Ablagekennung den explizit definierten Adresszonen zuordnen.

ABLAGE zone (k) zone: Name einer explizit vereinbarten Adresszone
 k: Ablagekennung
 K0 V0 B0 D0
 K1 V1 B1 D1
 K2 V2 B2 D2

Mehrere Angaben durch Komma trennen,

Alle auf einen Pseudobefehl ABLAGE folgenden Informationseinheiten, die eine Ablagekennung aufweisen, welche mit einer der im Pseudobefehl ABLAGE angegebenen Ablagekennungen übereinstimmt, werden der im Pseudobefehl ABLAGE angegebenen Adresszone zugeordnet. Dieser Pseudobefehl legt also für die unter k angegebene Ablagekennung eine neue Zuordnung fest. Für k können mehrere Ablagekennungen angegeben werden, die durch Komma zu trennen sind. Die Informationseinheiten werden in der Reihenfolge, in der sie im Quellenprogramm auftreten, in der angegebenen Zone abgelegt. Die Adresszone muß vor dem Pseudobefehl ABLAGE definiert sein.

Die durch den Pseudobefehl ABLAGE angegebene Zuordnung zu explizit definierten Adresszonen kann entweder durch einen neuen Pseudobefehl ABLAGE

oder durch den Pseudobefehl AEND ganz oder teilweise aufgehoben werden.

AEND (k) k: Ablagekennung
 K0 V0 B0 D0
 K1 V1 B1 D1
 K2 V2 B2 D2

Mehrere Angaben durch Komma trennen,

Alle folgenden Informationseinheiten mit derselben Ablagekennung, wie sie im Pseudobefehl AEND angegeben ist, werden wieder in der zugehörigen implizit definierten Adresszone abgelegt.

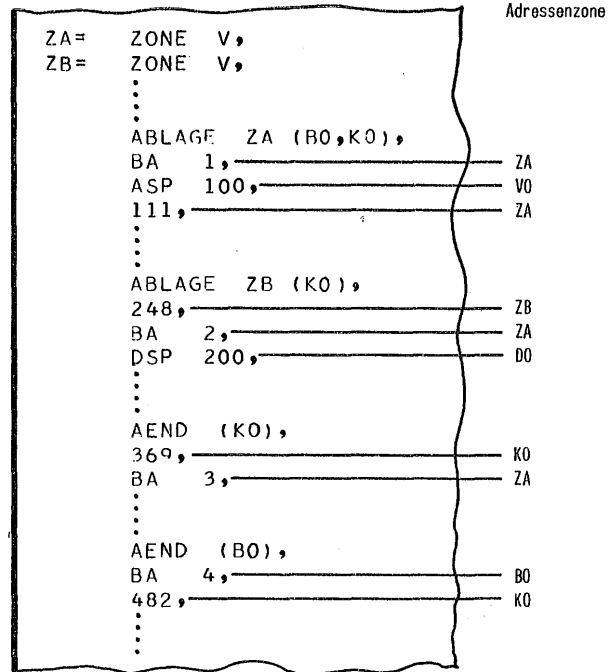


Bild 5.2 Beispiel für die Zuordnung zu explizit definierten Adresszonen

5.2. Anordnung von Adresszonen in Gebieten

Das Montageprogramm erstellt für die implizit und explizit definierten Adresszonen Gebiete. Über die Anordnung der explizit definierten Adresszonen in einem explizit definierten Gebiet kann aber auch mit dem Pseudobefehl ZONAN verfügt werden.

ZONAN gebiet (zone) gebiet: Gebietsname
ZONAN lücke (fzone) lücke: Lückenname

zone: Name der explizit definierten Adresszone, mehrere Angaben durch Komma trennen

fzone: Name der Freihaltezone, mehrere Angaben durch Komma trennen

In Gebieten dürfen keine Freihaltezonen angeordnet werden. Die Freihaltezonen werden in Lücken angeordnet. Die angegebenen Adresszonen werden in der Reihenfolge ihrer Niederschrift unmittelbar hintereinander in dem genannten Gebiet angeordnet.

Es können mehrere Pseudobefehle ZONAN mit dem gleichen Gebietsnamen auftreten. Die Adresszonen werden dann fortlaufend angeordnet.

Bei verschiedenen Adressierungsbedingungen der Zone und des Gebiets in bezug auf den Schreibschutz wird eine Warnung ausgegeben. Die Adressierungsbedingung der Zone wird dann abgewandelt. Dabei wird

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

abgeändert.

```
ZONAN GEB1 (Z1,Z2,Z3,Z4),
ZONAN G12 (AZONE,BZONE),
```

Bild 5.3 Beispiel für die Anordnung von Adresszonen in Gebieten

Alle in einem Pseudobefehl ZONAN auftretenden Zonen und das Gebiet bzw. die Lücke müssen vorher im Quellenprogramm deklariert sein (durch die Pseudobefehle ZONE, GEBIET bzw. LUECKE).

Die Lage des Gebiets im Adressenraum kann durch eine absolute Angabe oder durch eine Angabe der Reihenfolge festgelegt werden. Die absolute Angabe kann dezimal oder sedezimal erfolgen. Die Angabe der Reihenfolge geschieht durch Namen von Gebieten oder Lücken, die unmittelbar vor dem neuen Gebiet oder der Lücke im Adressenraum angeordnet sind. Die in Klammern genannten Gebiete und Lücken beginnen also bei einer absolut festgelegten Adresse oder im Anschluß an ein vorhergehendes Gebiet bzw. eine vorhergehende Lücke.

```
GEBAN G6 (G7,G8,G9),
GEBAN AG (BG,CG,DG),
```

Bild 5.4 Beispiel für die Anordnung von Gebieten im Adressenraum

Jedes Gebiet erhält einen zusammenhängenden Teil des Adressenraums. Dieser darf mit den Adressierungsbedingungen des Gebietes nicht im Widerspruch stehen, da sonst die Montage abgebrochen wird. Die Montage wird auch abgebrochen, wenn die Adressierungsbedingung einer Adresszone in einem Gebiet mit dem Adressenraum nicht vereinbar ist.

Alle in einem Pseudobefehl GEBAN genannten Gebiete und Lücken müssen vorher im Quellenprogramm deklariert sein (durch die Pseudobefehle GEBIET bzw. LUECKE).

G

5.3. Anordnung von Gebieten im Adressenraum

Explizit vereinbarte Gebiete und Lücken müssen explizit im Adressenraum angeordnet werden. Dies geschieht mit dem Pseudobefehl GEBAN.

GEBAN bezug (liste) bezug: Anfang der Gebietsanordnung
 im Adressenraum,
 Seitennummer dezimal
 oder sedezimal,
 Gebietsname oder Lückename

liste: Gebietsname,
 Lückename,

(mehrere Angaben durch Komma trennen)

Für die Querbezüge werden Namen verwendet, deren Geltungsbereich über das Quellenprogramm hinaus geht. Sie haben innerhalb des ganzen Operators Gültigkeit. Es sind dies die Montagenamen. Durch das UEBERSETZE-Kommando oder den ersten Pseudobefehl SEGM des Quellenprogramms (siehe 2.1) wird ein Montagenamen für das Montageobjekt festgelegt. Weitere Montagenamen können mit dem Pseudobefehl EINGG angegeben werden. Außerdem können Kontaktnamen festgelegt werden, die aber nur in Verbindung mit einem Montagenamen zu Querbezügen verwendet werden können. Ihr Geltungsbereich ist auf das Quellenprogramm beschränkt.

Um die Querbezüge zu ermöglichen, muß in jedem Quellenprogramm, das beim Montagevorgang benötigt wird, angegeben sein

- mit welchem Namen sich ein fremdes Programm auf Adressen des eigenen Programms beziehen kann (Eingang),
- mit welchem Namen sich das eigene Programm auf Adressen fremder Programme beziehen kann (Externbezug).

6.1. Eingang

Mit dem Pseudobefehl EINGG kann ein zusätzlicher eigener Montagenamen festgelegt werden.

EINGG mon mon: Zusätzlicher Montagenamen, auf den sich andere Quellenprogramme beziehen können

Auf diesen Montagenamen können sich andere Quellenprogramme beziehen. Der Montagenamen muß jedoch im eigenen Quellenprogramm als Name vereinbart sein (z. B. als Benennung einer Konstanten, eines Befehls oder eines Speicherbereichs).

Mit dem gleichen Pseudobefehl können auch Kontaktnamen vereinbart werden.

EINGG (ko) ko: Kontaktname oder Liste von Kontaktnamen (durch Komma getrennt)

Die Kontaktnamen müssen im eigenen Quellenprogramm als Namen vereinbart sein. Jedes andere Programm kann sich dann auf diese Kontaktnamen beziehen, wenn außer dem Kontaktnamen auch eine Montagenamen angegeben wird.

Ein zusätzlicher eigener Montagenamen (sogen. unechter Montagenamen) wird gleichzeitig auch Kontaktname.

```

    UEBERSETZE, SPRACHE=TAS, NUM.--STD-,
                MO=-STD-, QUELLE=/

    FRMO= SEGM,
           BA 1,
           .
           .
           EINGG (FRMO, D, E, F),
           .
           .
    
```

Bild 6.1 Beispiel für die Benutzung des eigenen Montagenamen als Kontaktname

6.2. Externbezug

Ein fremder Montagenamen kann mit dem Pseudobefehl EXTERN angegeben werden.

EXTERN mon mon: Montagenamen des fremden Quellenprogramms

Wird auf Namen Bezug genommen, die als Kontaktnamen vereinbart werden, so muß zusätzlich ein Montagenamen angegeben werden, da bei der Montage nur die Montagenamen, nicht aber die Kontaktnamen bekannt sind.

Der Pseudobefehl hat dann folgende Form

EXTERN mon (ko) mon: Montagenamen des fremden Quellenprogramms
 ko: Kontaktname des fremden Quellenprogramms oder Liste von Kontaktnamen (durch Komma getrennt)

Alle Bezüge zwischen Quellenprogrammen müssen vereinbart sein. Dabei müssen die Namen, die im Pseudobefehl EINGG stehen, im eigenen Programm definiert sein und die Namen, die im Pseudobefehl EXTERN stehen, im eigenen Programm nicht vereinbart sein.

Ein fremder Montagenamen kann nur dann wie der Name einer Kernspeicheradresse benutzt werden, wenn er beim Pseudobefehl EXTERN auch als Kontaktname aufgeführt wird oder wenn der Pseudobefehl EXTERN nur den fremden Montagenamen enthält. Der fremde Montagenamen muß hierzu im fremden Montageobjekt als Kontaktname vereinbart sein.

```

    EXTERN FRMO (FRMO, D, E, F),
    EXTERN ANTON,
    
```

Bild 6.2 Beispiel für die Benutzung des fremden Montagenamen als Kontaktname

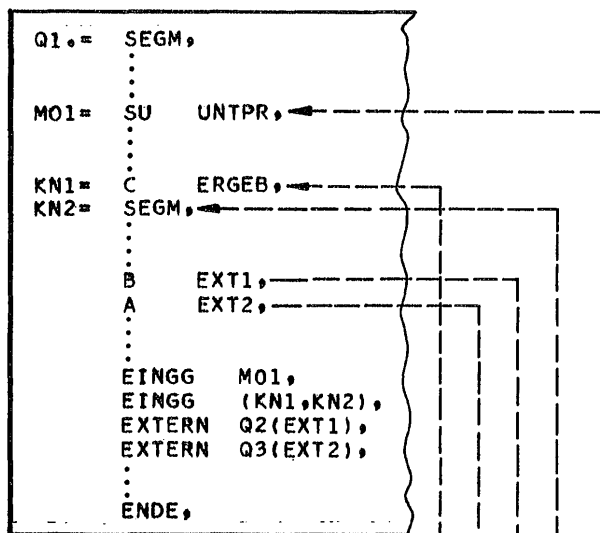
Neben dem Pseudobefehl EXTERN gibt es noch den Pseudobefehl EXTOPT.

EXTOPT mon mon: Montagename des fremden Quellenprogramms
 EXTOPT mon (ko) ko: Kontaktname des fremden Quellenprogramms oder Liste von Kontaktnamen (durch Komma getrennt)

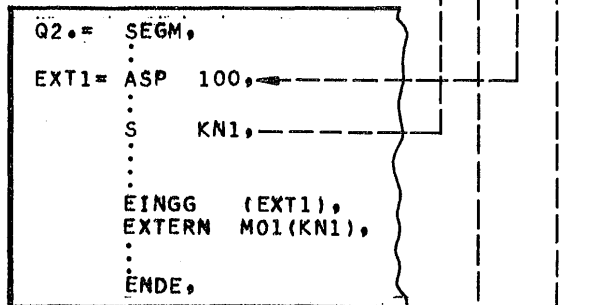
Es werden die gleichen Kontaktdeklarationen erzeugt wie beim Pseudobefehl EXTERN.

Im Gegensatz zum Pseudobefehl EXTERN muß jedoch das fremde Montageobjekt bei der Montage nicht zur Verfügung stehen, da es nicht notwendig anmontiert wird.

Quellenprogramm 1



Quellenprogramm 2



Quellenprogramm 3

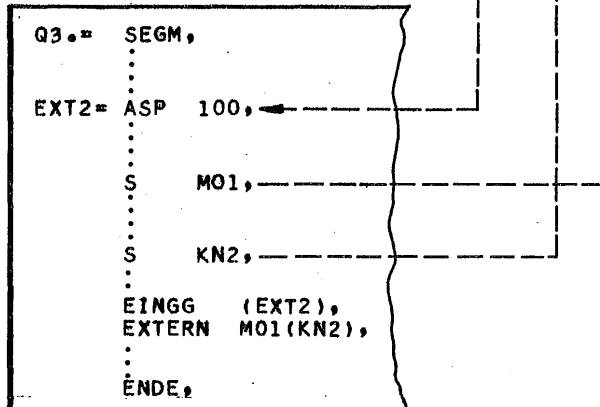


Bild 6.3 Beispiel für Querbezüge zwischen Quellenprogrammen

7. ZUORDNUNGEN

7.1. Zuordnung der Indexnamen

Jeder Indexzelle ist eine Indexadresse zugeordnet, mit der sie adressiert werden kann. Indexadressen können implizit vom Assembler oder explizit mit Hilfe des Pseudobefehls INDEX benannt werden.

INDEX p (x) p: Indexpegel, absolute oder symbolische Indexadresse oder leer
 x: Indexliste, symbolische Indexadressen, Summenbezüge und Freihaltezonen (0 bis 255) (durch Komma getrennt)

Mit dem Pseudobefehl INDEX können Indexadressen bestimmte globale und lokale Indexnamen explizit zugeordnet werden. Die Zuordnung beginnt bei dem angegebenen Indexpegel. Der Indexpegel kann nicht nur durch eine absolute Indexadresse (0 bis 255), sondern auch durch eine

symbolische Indexadresse angegeben werden. Im letzten Fall muß die Indexadresse in einem vorhergehenden Indexbefehl explizit vereinbart worden sein. Die Indexpegel-angabe ist nicht zwingend, sie darf fehlen. Dann wird die Indexliste vom Assembler als Verlängerung der im vorangehenden Pseudobefehl INDEX angegebenen Indexliste interpretiert. Fehlt der Indexpegel auch im ersten Pseudobefehl INDEX, dann beginnt die erste Indexliste mit der Adresse 0.

Die Indexliste kann lokale und globale Indexnamen und Freihaltezahlen enthalten. Der erste in der Liste enthaltene Indexname wird dem Indexpegel zugeordnet. Die anderen Indexnamen werden der Reihe nach den Indexadressen zugeordnet, die auf den Indexpegel folgen. Die globalen Indexnamen werden durch einen Punkt gekennzeichnet. Freihaltezahlen geben die Zahl der freizuhaltenden Indexadressen an, denen durch den aktuellen Pseudobefehl INDEX kein Name zugeordnet wird. Die Freihaltezahl kann Werte von 0 bis 255 annehmen.

Die explizit definierten Indexadressen liegen im Speicher vor den implizit definierten Indexadressen. Damit wird implizit eine Doppelbelegung von Indexzellen ausgeschlossen. Explizit ist eine Doppelbelegung durch Wahl geeigneter Indexpegel möglich. Näheres zu den Indexadressen ist im Abschnitt C 6 beschrieben.

Niederschrift

```

S1=  SEGM,
      .
      .
      .
      INDEX 2(X1,X2,X3),
      .
      .
      .
S2=  SEGM,
      .
      .
      .
      INDEX X2(5,X4,X5,X6),
      .
      .
      .
S3=  SEGM,
      .
      .
      .
      INDEX X2(2,X7),
      .
      .
      .
  
```

Zuordnung

absolute Adresse	symbolische Adresse
0	
1	
2	X1
3	X2
4	X3
5	X7
6	
7	
8	X4
9	X5
10	X6

Bild 7.1 Beispiel für explizite Indexbenennung

7.2. Vereinbarung von Oktadennamen

Mit Hilfe des Pseudobefehls TEXT können Oktaden mit Namen benannt werden. Diese Oktadennamen dürfen nach ihrer Vereinbarung in Textfolgen anstelle der Oktadenwerte stehen.

name = TEXT z name: Name der Oktade
 z: Wert der Oktade
 Zahl (0 ... 255) oder
 2 Tetraden ('..')

```

PUNKT=TEXT 169,
X=     TEXT 'D7',
MINUS=TEXT 145,
PI=    TEXT '7F',
  
```

Bild 7.2 Beispiele für Benennung von Oktaden

7.3. Gleichsetzung

Mit Hilfe des Pseudobefehls GLCH können beliebige Adressenteile oder bestimmte Teile von Adressenteilen mit einem Namen versehen werden. Die Namen können einen lokalen oder globalen Geltungsbereich haben.

```
name = GLCH b  name: Name des Adressenteils
                b: Ganz-, Rechts-, Links- oder Rechts-
                  und Linksadressenteil oder Teile
                  davon
```

Der Name darf in beliebigen Adressenteilen eines TAS-440-Befehls anstelle des benannten Ausdruckes stehen. Die Namen müssen so gewählt werden, daß eine Verwechslung mit anderen Ausdrücken in Adressenteilen ausgeschlossen ist.

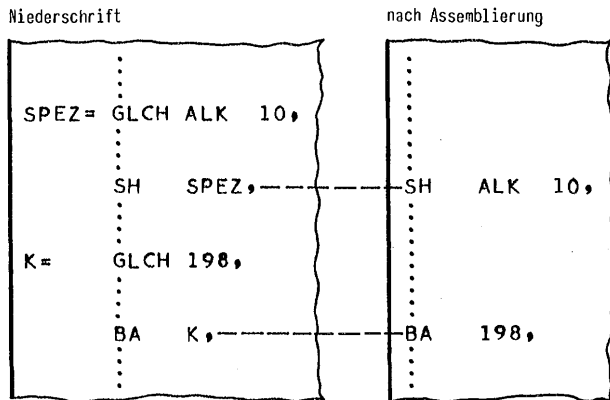


Bild 7.3 Beispiele für Gleichsetzung

Gleichgestzte Namen dürfen nicht im Adressenteil von Pseudobefehlen stehen.

Ausnahmen: GLCH, ASP und DSP. Dies gilt bei ASP und DSP nur unter der Voraussetzung, daß die gleichgesetzten Namen vorher definiert waren.

7.4. Codeumsteuerung für Oktadenfolgen

Der vom Assembler erzeugte Objektcode von Oktadenkonstanten und von Oktadenfolgen in Textkonstanten oder in Bitfeldkonstanten kann mit dem Pseudobefehl CODE in einem vom Zentralcode verschiedenen Code umgeschlüsselt werden.

```
CODE s  s: Codespezifikation
        D = umschlüssel in Druckercode DC2
        S = umschlüssel in Schreibmaschinencode SMC1
        Z = Zentralcode ZC1
```

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

Zu Beginn der TAS-Übersetzung ist implizit der Zentralcode ZC1 eingestellt. Der Zentralcode kann aber auch durch die Codespezifikation Z eingestellt werden.

Jeder Pseudobefehl CODE ist gültig bis zum nächsten oder bis zum Pseudobefehl ENDE.

7.5. Oktadenadressierung

Oktadenadressen, die für die Befehle ZK und TOK nötig sind, können mit Oktadenadrefkonstanten angegeben werden.

```
name = OA p b  name: Name der Oktadenadrefkonstanten
                p: Parameter 0 ... 255
                b: Adresse der Oktadenfolge
                  (KSP-Bezug)
```

Eine Oktadenadrefkonstante wird als Pseudobefehl OA mit zwei Adrefteilen geschrieben. Der Pseudobefehl darf benannt werden. Der Parameter kann Werte von 0 bis 255 annehmen. Er dient zur Angabe der Position des Bytes relativ zum ersten Byte des durch die Adresse der Oktadenfolge (KSP-Bezug) angegebenen Halbwords.

Als Montagecode erhält man eine Halbword-Wertbestimmung mit dem Wert $(3 \times b) + p$ und dem Translationschlüssel 4.

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

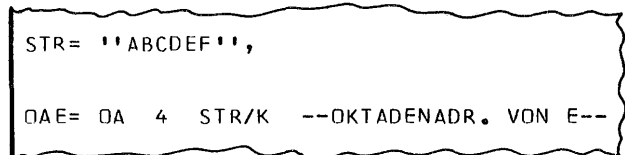


Bild 7.4 Beispiel für Oktadenadressierung

8. STEUERUNG DES PROTOKOLLDRUCKS

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

8.1. Umschalten des Protokollendrucks

Wird im UEBERSETZE-Kommando ein Protokoll des Übersetzungsvorgangs gefordert, so kann mit dem Pseudobefehl DRUCK eine Umschaltung des Protokollendrucks erreicht werden. Dem Assembler wird hierdurch angezeigt, was gedruckt werden soll.

DRUCK p p: Druckparameter
 0 = kein Protokoll
 1 = originaler Quelltext
 2 = originaler Quelltext
 und im Quelltext
 definierte Makros
 3 = gesamter Quelltext

Mit dem Druckparameter wird angegeben, welche Teile des Übersetzungsvorgangs ausgedruckt werden sollen. Es stehen vier Möglichkeiten zur Verfügung.

Der Pseudobefehl DRUCK wirkt nur auf die Protokollierung der folgenden Informationseinheiten. Seine Wirkung wird beendet durch einen neuen Pseudobefehl DRUCK oder durch den Pseudobefehl ENDE. Zu Beginn des Programms wird implizit der Pseudobefehl DRUCK 2 gegeben. Wenn durch das UEBERSETZE-Kommando kein Protokoll gefordert wird, dann sind die Pseudobefehle DRUCK wirkungslos; sie werden dann überlesen. Wird aber bei der Übersetzung von Befehlen und Konstanten ein Fehler erkannt, so wird die Fehlermeldung (z.B. Adressenteil fehlt, Name doppelt definiert usw.) immer protokolliert.

```
DR=  SFGM,  
     DRUCK 3,  
     .  
     .  
     DRUCK 0,  
     .  
     .  
     ENDE,
```

Bild 8.1 Beispiel für Umschalten des Protokollendrucks

8.2. Einfügen von Leerzeilen

Im Assemblerprotokoll können mit dem Pseudobefehl ZEILE Leerzeilen eingefügt werden.

ZEILE z z: Zeilenzahl, Zahl von 1 ... 6

Im Protokoll wird der Pseudobefehl ZEILE durch Leerzeilen ersetzt.

```
ZEILE 6,
```

Bild 8.2 Beispiel für Angabe von Leerzeilen

8.3. Unterdrückung "VERBOTENER BEZUG"

Die Fehlermeldung "VERBOTENER BEZUG" kann mit dem Pseudobefehl UVB unterdrückt werden.

UVB p p: Parameter
 0 = Ausschaltung der Wirkung
 1 = Einschaltung der Wirkung

Die Wirkung wird mit 1 für die folgenden Einheiten eingeschaltet und mit 0 wieder ausgeschaltet.

```
UVB 1,  
FIN=S  NZFI1/K --NAECHST ZEICH 1.ART--  
      S  NZFI2/K --NAECHST ZEICH 2.ART--  
      MU  S  0/K --EINGABE-ENDE--  
UVB 0,  
MF  XEIN,  
T   FIN,
```

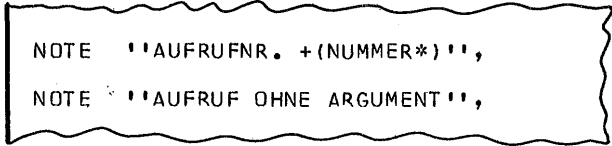
Bild 8.3 Beispiel für sinnvolle Anwendung

8.4. Oktadenstring als Kommentar

Mit dem Pseudobefehl NOTE können Oktaden im Protokoll als Kommentar ausgedruckt werden.

NOTE f f: Oktadenfolge

Falls der Assembler den Pseudobefehl NOTE interpretiert, er z. B. nicht in einer unterdrückten Versionsfolge liegt, wird die angegebene Oktadenfolge im Protokoll als Kommentar-Informationseinheit mit Pseudo-Fehlermeldung ausgedruckt.



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

Bild 8.4 Beispiel für Pseudobefehl NOTE

9. MAKROS UND WIEDERHOLUNGEN

Mit den Elementen der Makrosprache können Folgen von Informationseinheiten (Befehle, Pseudobefehle und Konstanten) an beliebiger Stelle eines TAS-Quellenprogramms eingesetzt werden. Zu diesem Zweck muß das Makro vereinbart werden. Dabei erhält es einen Namen, mit dem es im Quellenprogramm aufgerufen werden kann.

Bei der Vereinbarung können außerdem bestimmte Parameter (formale Parameter), für die erst beim Aufruf ein aktueller Wert (aktueller Parameter) angegeben wird, im Makrotext besetzt werden. Der Makrotext wird durch die Folge von Informationseinheiten innerhalb des Makros bestimmt (z.B. Text von DEF bis DEND).

Diese vorbereitete Folge kann jederzeit aufgerufen werden, wobei bei jedem Makroaufruf andere aktuelle Werte angegeben werden können. Dadurch wird der einmal definierte Text variabel.

Aktuelle und formale Parameter werden auch als Makrokonstante bzw. Makrovariable bezeichnet.

Ausführlich ist die Makrotechnik im Abschnitt "Programmteile, 7. Makrotechnik" beschrieben. Dort sind auch umfangreiche Beispiele zur praktischen Anwendung der Makrosprache angegeben.

9.1. Makrodefinition

Die Definition eines Makros hat folgende Form

DEF name	name: Name des Makros, unter dem es aufgerufen werden kann
text	text: Makrotext, der anstelle des Aufrufs eingesetzt werden soll
DEND	

Die Definition wird mit dem Pseudobefehl DEF eingeleitet und mit dem Pseudobefehl DEND abgeschlossen.

Das Makro erhält einen Namen, der im ganzen Quellenprogramm gültig ist. Mit dem Namen kann das Makro an jeder beliebigen Stelle im Quellenprogramm aufgerufen werden. Der Name des Makros muß die für Namen erlaubte Form haben. Er darf jedoch nicht mit anderen Namen, die allgemein gültig im Quellenprogramm sind, übereinstimmen (z. B. R&LIES, R&DRUCKE usw.).

Der Makrotext kann jede in der TAS-Sprache erlaubte Form haben. Jedoch müssen immer vollständige Informationseinheiten verwendet werden. Im Makrotext können Parameter eingesetzt werden, die erst beim Makroaufruf ihren aktuellen Wert erhalten. Sie werden deshalb bei der Makrodefinition zunächst durch einen Namen symbolisch dargestellt. Näheres über die Parameter ist im Abschnitt 9.3. beschrieben.

9.2. Makroaufruf

Ein Makro wird unter dem vereinbarten Namen aufgerufen. Der Name des Makros ist somit ein Pseudobefehl.

name name: bei der Definition vereinbarter Name des Makros

Anstelle des Namens setzt der Assembler den vereinbarten Makrotext. Danach wird der Makrotext übersetzt.

Bild 9.1 zeigt die allgemeine Form eines Makros. Nach der Assemblierung steht anstelle des Makroaufrufs im Programm der Makrotext.

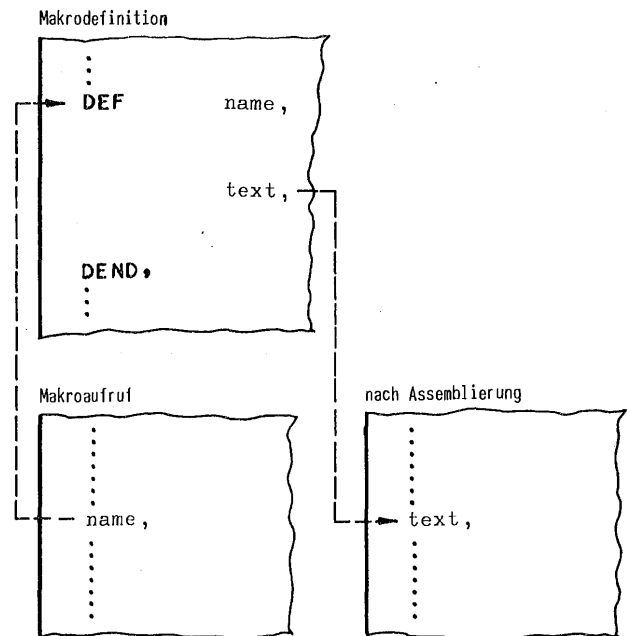


Bild 9.1 Allgemeine Form des Makros

9.3. Parameter

Makros können im einfachsten Fall einen Parameter haben. Dieser Parameter ist eine Größe, der bei jedem Makroaufruf ein anderer Wert zugewiesen werden kann. Dies ist der formale Parameter (Makrovariable). Beim Aufruf des Makros muß diesem Parameter der jeweils aktuelle Wert zugewiesen werden, der den aktuellen Parameter (Makrokonstante) darstellt.

Allgemeine Form des formalen Parameters:

+(name) name: Name des formalen Parameters

Dieser formale Parameter kann überall dort stehen, wo erst beim Aufruf des Makros dem Parameter ein Wert zugeordnet werden soll.

Der aktuelle Parameter darf enthalten:

- die Buchstaben A bis Z,
- die Ziffern 0 bis 9,
- die Sonderzeichen + Plus,
- Minus,
. Punkt,
* Stern,
CR Wagenrücklauf,
| Senkrechtstrich,
() runde Klammern paarweise,
' ' Apostroph paarweise.

Weiterhin darf ein aktueller Parameter sein:

- eine Tetradenfolge; durch Apostroph eingeschlossen,
- eine Oktadenfolge; durch Doppelapostroph eingeschlossen,
- eine Textfolge; mit der Zeichenfolge '(eingeleitet und mit der Zeichenfolge)' abgeschlossen,
- eine Liste von aktuellen Parametern, die in runde Klammern eingeschlossen sind.

Der aktuelle Parameter darf folgende Zeichen nicht enthalten:

- zwei aufeinanderfolgende Minuszeichen,
- , Komma,
- = Gleichheitszeichen,
- Space,
- () runde Klammern einzeln, (paarweise sind sie erlaubt),
- ' Apostroph einzeln, (paarweise ist er erlaubt),
- '' Doppelapostroph (paarweise erlaubt).

In Kommentaren und Überschriften ist ein Parameter nicht erlaubt. Bei Konstanten muß darauf geachtet werden, daß die Zeichenfolge +(nicht als Text auftaucht, sondern nur zur Einleitung eines formalen Parameters dient.

Beim Aufruf des Makros wird hinter dem Namen des Makros der aktuelle Parameter angegeben. Wird nur ein Parameter verwendet, so ist der Name des Makros gleichzeitig der Name des Parameters.

makro para makro: frei zu vereinbarender Makroname

para: aktueller Parameter für den angegebenen formalen Parameter

Im Bild 9.2. ist die allgemeine Form eines Makros mit einem Parameter dargestellt. Der Name des Makros ist gleichzeitig der Name des Parameters. Für das Makro ist der Name "name" vereinbart. Der Parameter "name" erscheint in der Definition an zwei Stellen. Das Makro wird zweimal aufgerufen. Hinter jedem Aufruf steht der jeweils aktuelle Wert des Parameters. In diesem Fall wert₁ und wert₂. Nach dem Assemblieren werden diese aktuellen Parameter für die formalen Parameter in den Makrotext eingesetzt.

Jedes Makro besitzt implizit nur einen Parameter. Sollen in einem Makro mehrere Parameter verwendet werden, so kann dieser eine implizite Parameter mit Hilfe des Pseudobefehls FORM in eine Parameterliste aufgliedert werden.

FORM p (u₁, u₂, ..., u_n)

p: Name des Parameters, der aufgliedert werden soll

u: Parameter der Parameterliste

Die Liste kann aus einem oder mehreren Namen für die formalen Parameter bestehen. Diese Namen können dann als formale Parameter im Makrotext benutzt werden.

Jeder formale Parameter in der Liste kann mit einem Wert vorbesetzt werden. Die Vorbesetzung hat die Form

name = wert name: Name des Parameters

wert: Wert, den der formale Parameter durch den aktuellen Parameter erhalten soll

Wenn beim Makroaufruf dem formalen Parameter kein aktueller Wert zugeordnet wird, dann wird der Wert der Vorbesetzung vom Assembler eingesetzt.

Beim Makroaufruf müssen die aktuellen Parameter in der Reihenfolge angegeben werden, wie die formalen definiert wurden. Der Aufruf hat dann die Form

name (werte) name: Name des Makros

werte: Liste der Werte, die die aktuellen Parameter dieses Aufrufs sind, durch Komma getrennt

Wird die Reihenfolge nicht eingehalten, so muß vor den aktuellen Wert der Name des formalen Parameters mit Gleichheitszeichen gesetzt werden.

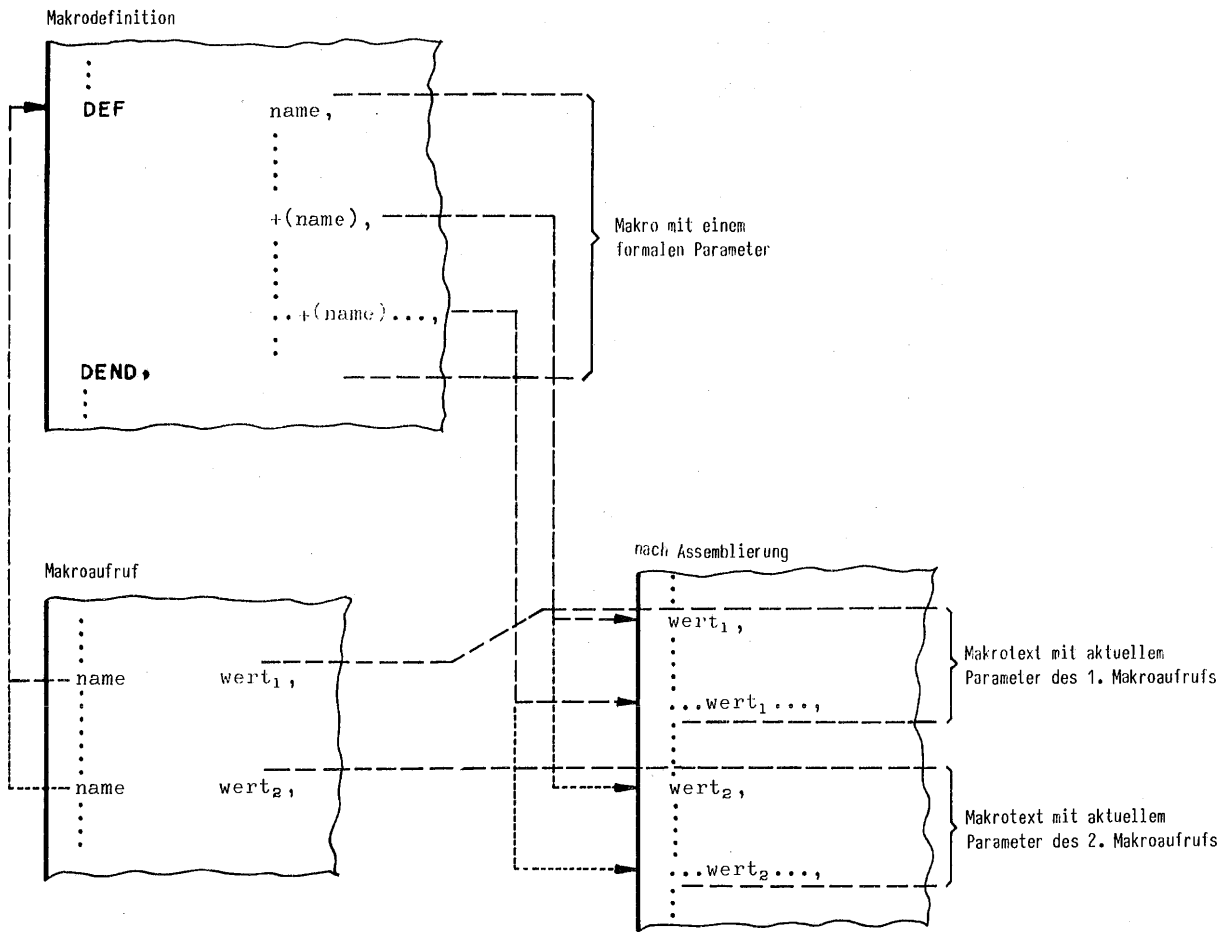


Bild 9.2 Allgemeine Form des Makros mit einem Parameter

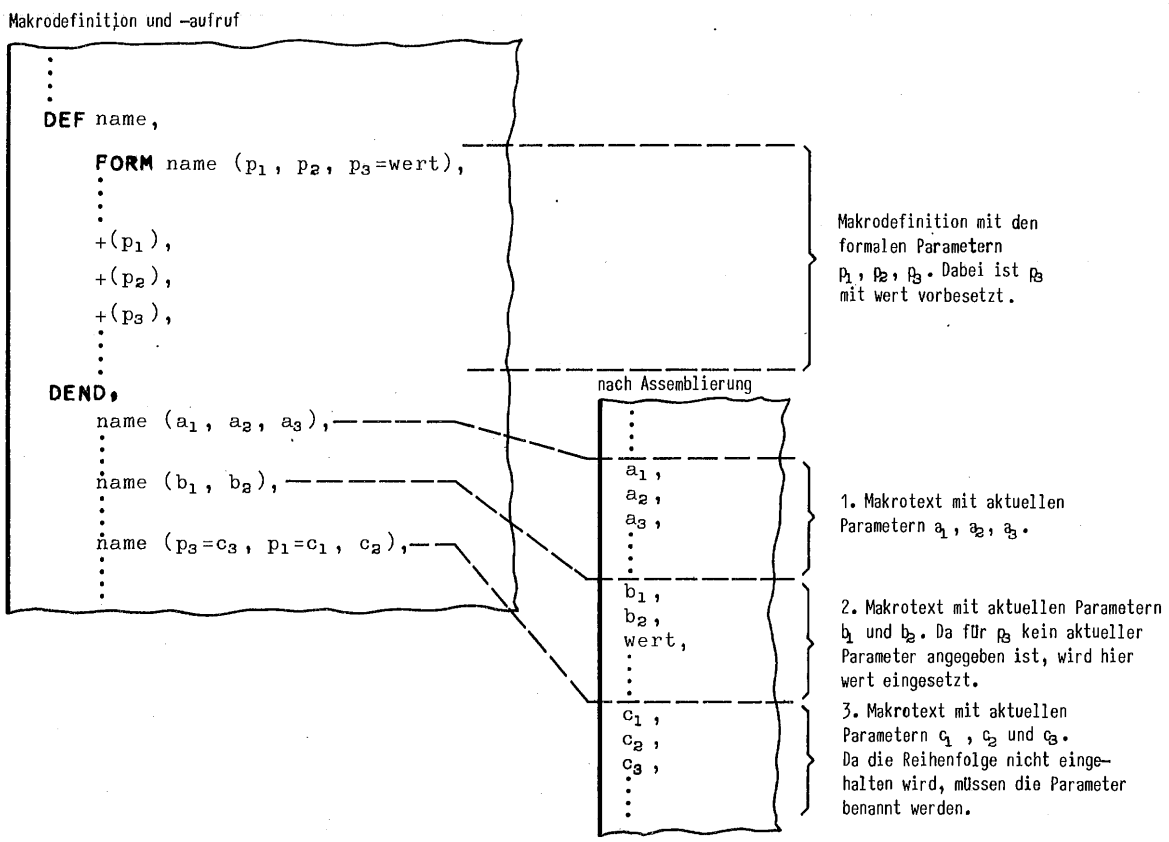


Bild 9.3 Allgemeine Form des Makros mit einer Parameterliste

TAS - Handbuch

April 71

G

Bild 9.3 zeigt die allgemeine Form des Makros mit einer Parameterliste. Der formale Parameter "name" wurde aufgegliedert in die formalen Parameter p_1 , p_2 und p_3 . Beim zweiten Makroaufruf ist für den formalen Parameter p_3 kein Wert in der Parameterliste angegeben. Deshalb wird hier vom Assembler der Wert der Vorbesetzung "wert" eingesetzt. Im dritten Makroaufruf ist die Reihenfolge der aktuellen Parameter nicht eingehalten worden. Darum muß die Abweichung $p_3 = c_3$ benannt werden. Auch der folgende Parameter, bei dem die ursprüngliche Reihenfolge beginnt, muß in diesem Fall benannt werden.

Jeder einmal im Makro definierte formale Parameter kann wiederum mit dem Pseudobefehl FORM in mehrere Parameter aufgegliedert werden. Die Parameterliste erhält dann den Namen dieses Parameters.

Außerdem gibt es formale Parameter, die im gesamten Quellenprogramm gelten. Es sind die globalen Parameter VERSION*, NUMMER*, DATUM* und GENV*. Näheres hierzu ist im Abschnitt "Programmteile, 7. Makrotechnik" beschrieben.

9.4. Versionen

Versionen können mit den Pseudobefehlen VERS, SONST und VEND vereinbart werden, wenn Texte nur unter bestimmten Bedingungen übersetzt werden sollen (Versionstexte). Die Texte für Versionen müssen die gleichen Bedingungen erfüllen wie die Makrotexe. Versionen können in Makros und Wiederholungen eingesetzt werden.

```

VERS (f = a)      f: Name des formalen Parameters
                 a: Wert des aktuellen Parameters
text
VEND             text: Versionstext, der nur übersetzt wird,
                 wenn die Bedingung f = a erfüllt ist

```

Hinter dem Pseudobefehl VERS wird in Klammern die Bedingung angegeben, unter der der nachfolgende Versionstext übersetzt wird. Abgeschlossen wird die Version durch den Pseudobefehl VEND.

Hinter dem Namen des formalen Parameters wird mit Gleichheitszeichen der Wert angegeben, den der aktuelle Parameter haben soll, damit der nachfolgende Versionstext übersetzt wird.

Es können auch mehrere Bedingungen angegeben werden (Bedingungsliste), die durch Komma zu trennen sind. Wenn eine dieser Bedingungen erfüllt ist, wird der folgende Versionstext übersetzt. Mehrere Bedingungen sind also durch die Funktion ODER (Disjunktion) verbunden.

Es gibt noch eine weitere Möglichkeit, die Bedingung bei einer Versionsanweisung anzugeben. Wenn einem formalen Parameter überhaupt irgend ein Wert zugewiesen wurde, wird der folgende Versionstext übersetzt. In diesem Fall braucht als Bedingung nur der Name des formalen Parameters angegeben zu werden.

```

VERS (f)          f: Name des formalen Parameters
                 text: Versionstext, der nur übersetzt wird,
VEND             wenn überhaupt irgend ein Wert für
                 den formalen Parameter angegeben wurde

```

Makrodefinition und -aufruf

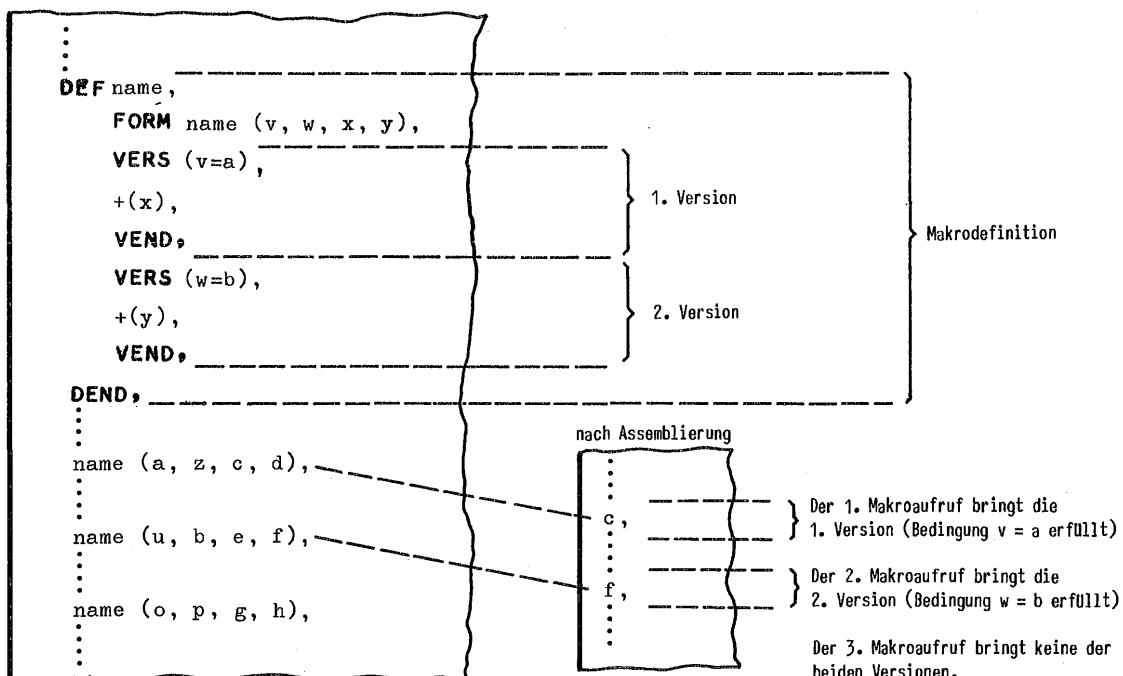


Bild 9.4 Allgemeine Form des Makros mit Versionen

Der Versionstext darf weitere Versionstext- (verschachtelte) oder Wiederholungs- und Makrotextdefinitionen enthalten. Sie müssen jedoch alle im Versionstext erscheinen. Ist keine der Bedingungen erfüllt, so wird der Versionstext überlesen. Der Assembler untersucht in diesem Fall nur die folgenden VERS- und VEND-Befehle, um das Ende des Versionstextes feststellen zu können. Wenn der Versionstext interpretiert wird, werden alle darin enthaltenen Pseudobefehle normal ausgeführt und im Versionstext enthaltene weitere Versionstexte normal interpretiert (damit lassen sich UND-Verknüpfungen von Bedingungen durch Verschachtelungen von Versionstextdefinitionen und ODER-Verknüpfungen durch Zusammenfassung von Bedingungslisten realisieren).

Zusätzliche Möglichkeiten ergibt die Anwendung von Bedingungsoperatoren (siehe Bild 9.5). Dadurch kann man auf Gleichheit oder Ungleichheit oder auf die Bedingung "kleiner (gleich)" oder "größer (gleich)" von entweder linksbündig oder rechtsbündig gedachten Zeichenfolgen prüfen:

- GL = gleich
- UG = ungleich
- LKG = linksbündig kleiner/gleich
- LKL = linksbündig kleiner
- LGG = linksbündig größer/gleich
- LGR = linksbündig größer
- RKG = rechtsbündig kleiner/gleich
- RKL = rechtsbündig kleiner
- RGG = rechtsbündig größer/gleich
- RGR = rechtsbündig größer

In Bild 9.4 ist ein Makro mit Versionstext in allgemeiner Form dargestellt. Die Bedingung der ersten Version ist beim ersten Aufruf erfüllt. Deshalb wird hier vom Assembler der Versionstext der ersten Version eingesetzt. Beim zweiten Aufruf ist die Bedingung der zweiten Version erfüllt. Beim dritten Aufruf ist keine der beiden Bedingungen erfüllt. Deshalb werden hier beide Versionstexte nicht übersetzt.

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

G

TAS - Handbuch

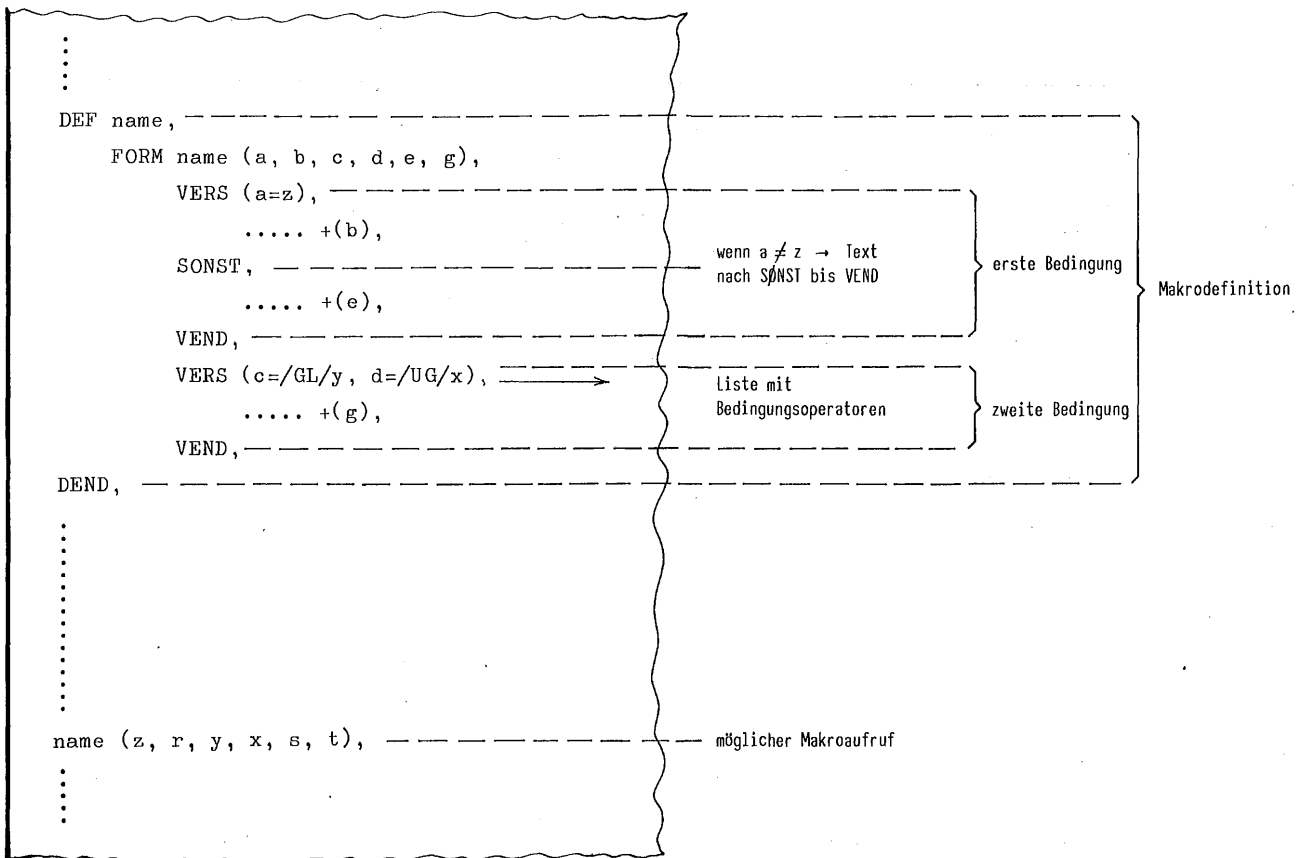


Bild 9.5 Allgemeine Form für eine Versionstextdefinition mit Bedingungsoperatoren

April 71

9.5. Wiederholungen

Mit Hilfe des Pseudobefehls WIED können Teile eines Programms wiederholt werden. Die Texte für Wiederholungen müssen die gleichen Bedingungen erfüllen wie die Makrotexte. Die Texte müssen mindestens einen formalen Parameter haben.

```

para = WIED (liste)   para: Name des formalen Parameters
                      liste: Liste des aktuellen Parameter
                        für die Wiederholungen, durch
                        Komma getrennt
                      text: Wiederholungstext
WEND
  
```

Der auf den Pseudobefehl WIED folgende Text wird wiederholt. Abgeschlossen wird diese Anweisung durch den Pseudobefehl WEND.

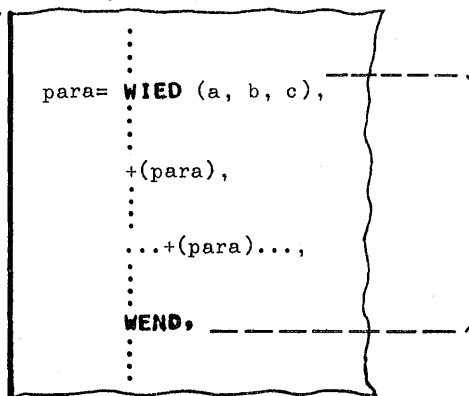
Vor dem Pseudobefehl WIED wird der Name des formalen Parameters angegeben, der im Wiederholungstext verwendet wird. Für den formalen Parameter gelten die gleichen Vorschriften, wie bei den Makros.

Hinter dem Pseudobefehl WIED werden in Klammern die aktuellen Parameter für die einzelnen Wiederholungen angegeben. Der Text wird so oft wiederholt, wie aktuelle Parameter vorhanden sind. Bei jedem Durchlauf des Wiederholungstextes wird immer ein aktueller Parameter nach dem anderen verwendet.

Im Bild 9.6 ist die allgemeine Form einer Wiederholungsanweisung dargestellt. Die Wiederholungsanweisung enthält drei aktuelle Parameter. Der Text wird deshalb dreimal wiederholt. Bei jeder Wiederholung wird ein aktueller Parameter in den Text eingesetzt.

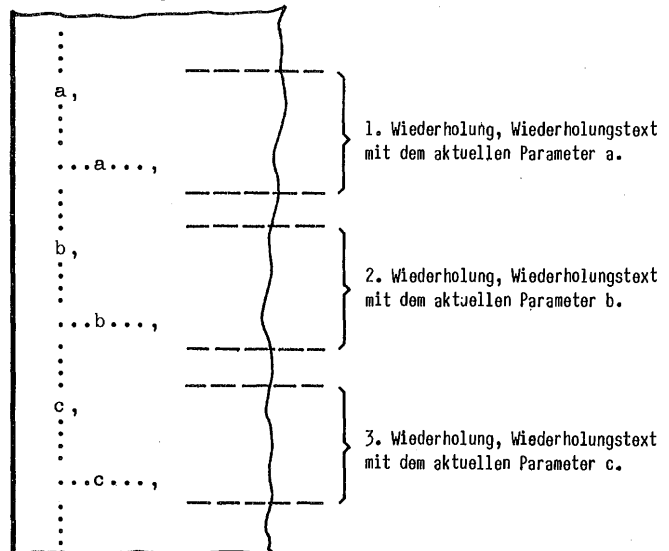
In Wiederholungen können auch Makros und Versionen auftreten. Es können aber auch Wiederholungen in Makros stehen.

Wiederholungsanweisung



Wiederholungsanweisung mit dem formalen Parameter para und den aktuellen Parametern a, b und c.

nach Assemblierung



1. Wiederholung, Wiederholungstext mit dem aktuellen Parameter a.

2. Wiederholung, Wiederholungstext mit dem aktuellen Parameter b.

3. Wiederholung, Wiederholungstext mit dem aktuellen Parameter c.

Bild 9.6 Allgemeine Form einer Wiederholungsanweisung

TELEFUNKEN
COMPUTER



TAS - HANDBUCH
Festkomma-Arithmetik

INHALT



FESTKOMMA-ARITHMETIK

<u>1.</u>	<u>EXTERNE DARSTELLUNG</u>	steht noch aus
<u>2.</u>	<u>INTERNER AUFBAU</u>	2 - 1
2.1.	Bereich	2 - 1
2.2.	Vorzeichen	2 - 1
2.3.	Bereichüberschreitung	2 - 2
2.4.	Marke	2 - 2
2.4.1.	Setzen der Marke	2 - 3
2.4.2.	Löschen der Marke	2 - 3
2.4.3.	Abfragen der Marke	2 - 4
2.5.	Typenkennung	2 - 4
2.6.	Einerrücklauf	2 - 4
2.7.	Null als Ergebnis	2 - 4
2.8.	Betrag	2 - 5
2.9.	Rundung	2 - 5
2.10.	Komma	2 - 5
2.10.1.	Komma links, rechts	2 - 5
2.10.2.	Komma beliebig	2 - 5
2.10.3.	Kommaverschiebung	2 - 5
2.11.	Vergleiche bei Sprüngen	2 - 6
<u>3.</u>	<u>OPERATIONEN FÜR EINFACHE WORTLÄNGE</u>	3 - 1
3.1.	Bringen	3 - 1
3.2.	Speichern	3 - 3
3.3.	Addition - Subtraktion	3 - 5
3.4.	Multiplikation	3 - 6
3.4.1.	Ergebnis mit doppelter Wortlänge	3 - 6
3.4.2.	Ergebnis mit einfacher Wortlänge	3 - 7
3.4.3.	Kommarechnung	3 - 7
3.5.	Division	3 - 7
3.6.	Schiften	3 - 12
3.7.	Umwandlung in eine Gleitkommazahl	3 - 12
<u>4.</u>	<u>OPERATIONEN FÜR DOPPELTE WORTLÄNGE</u>	4 - 1
4.1.	Vorzeichenangleich	4 - 1
4.2.	Verlängerung und Verkürzung	4 - 1
4.3.	Transporte	4 - 1
4.4.	Addition - Subtraktion	4 - 2
4.5.	Multiplikation	4 - 2
4.6.	Division	4 - 2
4.7.	Schiften	4 - 3
<u>5.</u>	<u>OPERATIONEN FÜR HALBE WORTLÄNGE</u>	5 - 1
5.1.	Transport	5 - 1
5.2.	Addition - Subtraktion	5 - 1
5.3.	Multiplikation	5 - 1
<u>6.</u>	<u>TEILWÖRTER</u>	6 - 1
<u>7.</u>	<u>KONVERTIERUNG</u>	steht noch aus



2. INTERNER AUFBAU

In der binären Darstellungsweise im Rechner können Festkommazahlen dargestellt werden mit

- einfacher Wortlänge: 46 Bits + 2 Bits Vorzeichen
- doppelter Wortlänge: 92 Bits + 4 Bits Vorzeichen
- halber Wortlänge: 22 Bits + 2 Bits Vorzeichen
oder 24 Bits = stets pos. Zahl

In den Abschnitten 3, 4 und 5 sind die Operationen für die verschieden langen Zahlen dargestellt.

2.1. Bereich

Bei einfacher Wortlänge stehen für die Zahl 46 Bits und für das Vorzeichen 2 Bits zur Verfügung. Handelt es sich um ganze Zahlen - das Komma steht in diesem Fall rechtsbündig - so können alle Zahlen von

$$\begin{aligned} & 0 \text{ bis } \pm (2^{46} - 1) \\ & = 0 \text{ bis } \pm 70\,368\,744\,177\,663 \\ & \approx 0 \text{ bis } \pm 70 \text{ Billionen} \end{aligned}$$

dargestellt werden. Für Größen, die das Vielfache einer Grundeinheit sind - z.B. Stückzahlen, Geldbeträge in Pfennigen, Gewichte in Gramm - werden im allgemeinen ganze Zahlen verwendet.

Für Größen, die Teile einer Einheit sind - z.B. Prozentzahlen (1% von 1 = 0,01) - werden die Festkommazahlen als echte Brüche dargestellt. Sie sind dann in jedem Falle kleiner als 1. Die kleinste darstellbare Einheit ist dann

$$\begin{aligned} & \pm 2^{-46} \\ & = \pm 0,000\,000\,000\,000\,014\,210 \dots \\ & = \pm 0,142\,108 \dots \cdot 10^{-13} \\ & \approx \pm 1/70 \text{ Billionen} \end{aligned}$$

Die Festkommazahl kann jedes Vielfache dieser kleinsten Einheit bis zum maximalen Wert von $1 - 2^{-46}$ annehmen. Das entspricht einem Zahlenbereich von

$$0 \text{ bis } \pm (1 - 2^{-46})$$

Das Komma kann an jeder beliebigen Stelle der Festkommazahl stehen, jedoch ist in diesem Fall eine Mitrechnung des Kommas notwendig. Hierauf ist näher im Abschnitt 2.10. eingegangen.

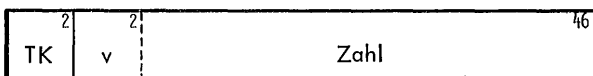


Bild 2.1 Festkommazahl einfacher Wortlänge

Bei doppelter Wortlänge stehen 92 Bits für die Zahl und 2×2 Bits für das Vorzeichen zur Verfügung. Bei ganzen Zahlen kann der Bereich

$$\begin{aligned} & 0 \text{ bis } \pm (2^{92} - 1) \\ & \approx 0 \text{ bis } \pm 4,95 \cdot 10^{27} \end{aligned}$$

überstrichen werden. Bei echten Brüchen beträgt der Bereich

$$0 \text{ bis } \pm (1 - 2^{-92})$$

Die kleinste darstellbare Einheit ist

$$\begin{aligned} & 2^{-92} \\ & \approx 0,2 \cdot 10^{-27} \end{aligned}$$

Die doppelte Wortlänge wird bei besonders großen Zahlen und zur Erzielung einer höheren Genauigkeit verwendet. Außerdem ist sie für Zwischenergebnisse bei der Multiplikation und Division erforderlich.

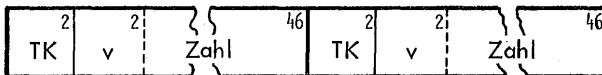


Bild 2.2 Festkommazahl doppelter Wortlänge

Bei Festkommazahlen mit halber Wortlänge gibt es zwei Möglichkeiten: 1. Zahlen mit Vorzeichen und 2. Zahlen ohne Vorzeichen (die dann stets positiv sind). Zahlen mit Vorzeichen können folgenden Bereich überstreichen:

Komma rechts	Komma links
$0 \text{ bis } \pm (2^{22} - 1)$	$0 \text{ bis } \pm (1 - 2^{-22})$
$= 0 \text{ bis } \pm 4\,194\,303$	$= 0 \text{ bis } \pm (1 - 0,000\,000\,23)$
$\approx 0 \text{ bis } \pm 4 \text{ Millionen}$	$\approx 0 \text{ bis } \pm 0,999\,999\,75$

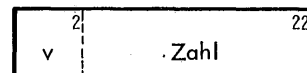
Kleinste Einheit: $2^{-22} \approx 1/4 \text{ Millionen}$

Bei den Zahlen ohne Vorzeichen können nur ganze positive Zahlen verwendet werden. Sie können den Bereich

$$\begin{aligned} & 0 \text{ bis } + (2^{24} - 1) \\ & = 0 \text{ bis } + 16\,777\,215 \\ & \approx 0 \text{ bis } + 16 \text{ Millionen} \end{aligned}$$

überstreichen. Wird das Komma nicht rechts angenommen, so ist eine zusätzliche Kommaerkennung erforderlich (siehe Abschnitt 2.10.).

mit Vorzeichen



nur positive Zahl

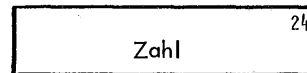


Bild 2.3 Festkommazahl halber Wortlänge

2.2. Vorzeichen

Bei Zahlen mit Vorzeichen wird das Vorzeichen in den linken beiden Bits dargestellt. Dabei bedeutet

- 00... = positive Zahl
- 11... = negative Zahl

Die beiden Bits müssen gleich sein. Im anderen Fall ist der zulässige Zahlenbereich über- bzw. unterschritten. Das linke Bit ist dann das Vorzeichen.

- 0L... = positive übergelaufene Zahl
- 1L... = negative untergelaufene Zahl

Der Über- bzw. Unterlauf ist in Abschnitt 2.3. beschrieben.

Eine positive Zahl wird in eine negative Zahl verwandelt, indem jedes Bit seinen umgekehrten Wert erhält - die Zahl wird also invertiert (B-1-Komplement).

Beispiel:

00 L00L = +9
 11 O110 = -9

Eine Subtraktion wird immer so durchgeführt, daß der Subtrahend invertiert und anschließend addiert wird. In diesem Zusammenhang kann sich eine positive (00 0000) oder eine negative (11 1111) Null ergeben. Eine negative Null hat nur interne Bedeutung und tritt beim Rechnen nach außen hin nicht in Erscheinung. Bei speziellen mathematischen Operationen ist es jedoch von Bedeutung, daß beim Ergebnis Null oft die negative Null entsteht.

Wird eine Festkommazahl halber Wortlänge mit Vorzeichen (rechtsbündig) in ein Rechenwerksregister gebracht, so werden die linken 24 Bits des Registers dem Vorzeichen des Halbwortes angeglichen. Damit steht die Zahl mit ihrem richtigen Wert im Register. Zahlen ohne Vorzeichen sind stets positiv, d.h. sie werden im Register links mit "Nullen" aufgefüllt.

Beispiel:

positives Vorzeichen	negatives Vorzeichen
00 L00L = +9	11 O110 = -9
00 0000L00L = +9	11 1111O110 = -9

ohne Vorzeichen
 L00L = +9
 00 0000L00L = +9

Bei Festkommazahlen doppelter Wortlänge werden die beiden Teile der doppelt langen Festkommazahl getrennt verarbeitet. Nach Anwendung der Befehle AQ oder SBQ können die Ergebnisse in beiden Registern verschiedene Vorzeichen haben. Obwohl Festkommazahlen doppelter Wortlänge auch dann von den arithmetischen Befehlen für doppelte Wortlänge normal verarbeitet werden, wenn die beiden Registerinhalte verschiedene Vorzeichen aufweisen, ist es doch zweckmäßig, vor Ausführung des nächsten Befehls die Vorzeichen mit Hilfe des Befehls VAQ einander anzugleichen.

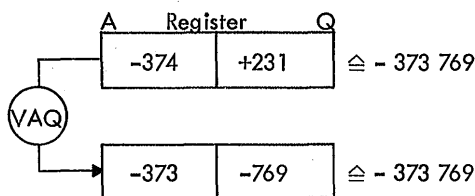


Bild 2.4 Vorzeichenangleich bei Festkommazahl doppelter Wortlänge

Beim Schiften einer Festkommazahl mit doppelter Wortlänge werden die Vorzeichenstellen des zweiten (rechten) Registers umschifft. In diesem Fall ist z.B. eine Vorzeichengleichheit stets erforderlich.

2.3. Bereichsüberschreitung

Wird der zulässige Zahlenbereich (siehe Abschnitt 2.2.) über- oder unterschritten, so sind die beiden Vorzeichenstellen nicht mehr identisch.

0L... = übergelaufene positive Zahl
 L0... = untergelaufene negative Zahl

Im allgemeinen erfolgt in diesen Fällen die Meldung "BÜ-Alarm". Diese Meldung wird vorerst jedoch vom Rechenwerk nur festgehalten und führt nicht zu einer Unterbrechung.

Sind Über- oder Unterläufe zu erwarten, so ist es zweckmäßig, mit dem Befehl SAA - Springe wenn arithmetischer Alarm (BÜ-Alarm) - eine Abfrage nach dem Über- oder Unterlauf vorzusehen. Im Alarmfall kann dann an eine andere Stelle des Programms gesprungen werden, um dort das Entsprechende zu veranlassen. Gleichzeitig wird die Alarmmeldung gelöscht. Soll der Über- oder Unterlauf ignoriert werden, so besteht die Möglichkeit, mit dem Befehl SAA auf die nächste Zeile zu springen. Der Befehl SAA hat dann nur das Löschen des BÜ-Alarms zur Folge.

Wird ein BÜ-Alarm nicht gelöscht, so können nur die Befehle, die das Befehlswerk benutzen, ausgeführt werden. Erst in dem Augenblick, in dem ein Befehl das Rechenwerk benutzen will, erfolgt (vor der Ausführung des Befehls) ein Eingriff mit Fehlermeldung "BÜ-Alarm". Weiteres siehe Abschnitt Fehlerbehandlung.

Eine Reihe von Befehlen verarbeitet auch Über- oder untergelaufene Zahlen. In diesen Fällen kann jedoch ein falsches Ergebnis bei gleichzeitigem Ausbleiben eines BÜ-Alarms entstehen. Deshalb ist es ratsam, der Einfachheit halber nicht mit über- bzw. untergelaufenen Zahlen zu rechnen. Nähere Einzelheiten sind aus den Beschreibungen zu den einzelnen Befehlen ersichtlich.

2.4. Marke

Alle Zahlwörter können im Speicher mit einer Marke versehen werden. Ein Zahlwort bezeichnet man als markiert, wenn das erste Bit auf L gesetzt ist. Beim Transport eines markierten Zahlwortes in eines der Rechenwerksregister wird das Markenregister auf L gesetzt, sofern es zuvor auf 0 stand. Wird dagegen ein nicht markiertes Wort in ein Rechenwerksregister gebracht, so bleibt das Markenregister unverändert. Das heißt mit anderen Worten: War das Markenregister bereits vorher auf L gesetzt, dann bleibt es L, ob nun ein markiertes oder ein unmarkiertes Zahlwort ins Rechenwerk transportiert wird. Bei Operationen, in denen eine Marke ausgewertet wird, muß zweckmäßigerweise das Markenregister zu Beginn des Programms mit einem entsprechenden Befehl (z.B. LA M) gelöscht (auf 0 gesetzt) werden.

$\langle M \rangle$	$:=$	$\langle M \rangle$	\vee	$\langle n \rangle_n$
0		0		0
L		0		L
L		L		0
L		L		L

Dieser Vorgang ist bei der Beschreibung der Wirkung der Befehle durch die Formel $\langle M \rangle := \langle M \rangle \vee \langle n \rangle_n$ ausgedrückt. Ist diese Formel nicht angegeben, wird das Wort unverändert übertragen, und der vorstehend beschriebene Vorgang findet nicht statt.

Bei doppelt langen Wörtern bewirken die Befehle BZ und BZN, daß das Markenregister auf L gesetzt wird, wenn eines der Wörter oder beide markiert sind ($\langle M \rangle := \langle M \rangle \vee \langle n \rangle_n \vee \langle n+2 \rangle_n$).

Gleichzeitig mit dem Transport des Wortes und dem evtl. Setzen des Markenregisters wird das erste Bit des Zahlwortes dem zweiten Bit angeglichen. Damit sind die beiden Vorzeichenbits gleich.

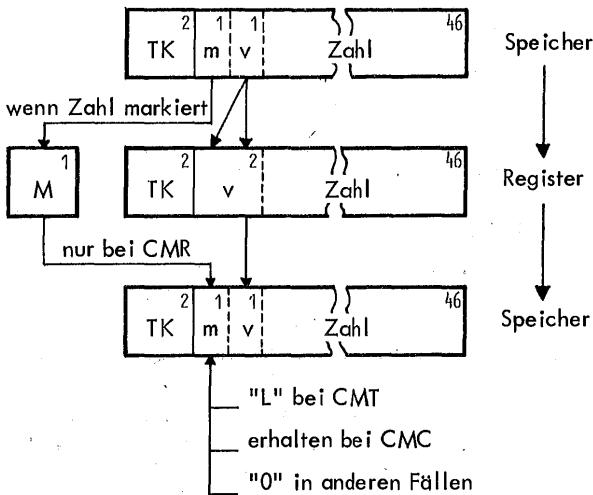
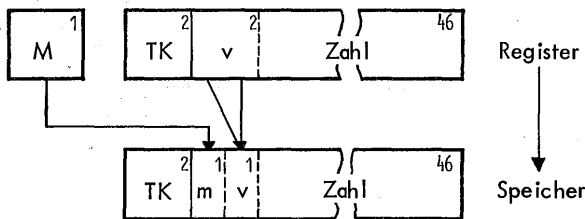


Bild 2.5 Transport der Marke

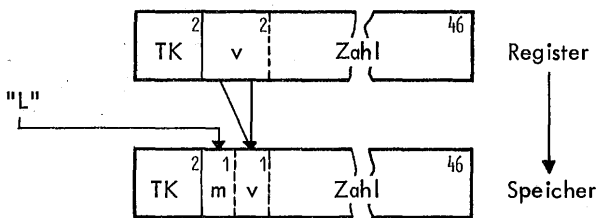
Im allgemeinen wird beim Abspeichern eines Zahlwortes aus dem Register das erste Bit auf Null gesetzt, das Wort wird also nicht markiert.

Folgende Befehle bilden eine Ausnahme:

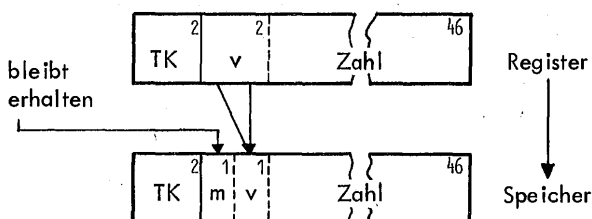
CMR Der Inhalt des Markenregisters wird in das erste Bit abgespeichert.*



CMT Das Zahlwort wird beim Abspeichern markiert (1. Bit auf "L" gesetzt).*

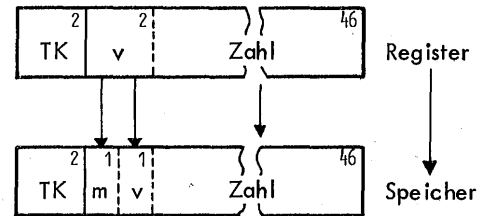


CMC Das 1. Bit in der Speicherzelle wird nicht verändert. Stand also in der Speicherzelle ein markiertes Wort, so ist auch das neue markiert.*



* Beide Vorzeichenbits des im Register stehenden Zahlwortes müssen gleich sein.

CU, C2, QCR Diese Befehle speichern unverändert, d.h. so wie im Register das entsprechende Bit gesetzt ist, wird auch das 1. Bit im Speicher gesetzt.



2.4.1. Setzen der Marke

Im Speicher kann ein Zahlwort markiert werden durch:

ZMC	n	Setze Marke im Speicher	$\langle n \rangle_m := L$
-----	---	-------------------------	----------------------------

In der Speicherzelle n wird das Markenbit $\langle n \rangle_m$ (1. Bit) auf L gesetzt.

GMT	n	Speichere markiert	$\langle n \rangle_m := L$ $\langle n \rangle := \langle A \rangle$
-----	---	--------------------	--

In der Speicherzelle n wird beim Abspeichern des Registers A das Markenbit auf L gesetzt.

LMT	n	Lösche markiert	$\langle n \rangle := +0$ $\langle n \rangle_m := L$
-----	---	-----------------	---

Die Speicherzelle n wird gelöscht und das Markenbit auf L gesetzt.

Das Markenregister M kann gesetzt werden durch:

ZTR	M	Setze Typenkennung im Register	$\langle M \rangle := L$
-----	---	--------------------------------	--------------------------

Ist bei dem Befehl ZTR als Spezifikation M angegeben, so wird das Register M auf L gesetzt. Danach besteht die Möglichkeit, mit dem Befehl CMR das Markenregister abzuspeichern.

2.4.2. Löschen der Marke

Im Speicher kann die Marke gelöscht werden durch:

LMC	n	Lösche Marke im Speicher	$\langle n \rangle_m := 0$
-----	---	--------------------------	----------------------------

Das Markenbit in der Speicherzelle n wird auf 0 gelöscht.

LC	n	Lösche Speicher	$\langle n \rangle := +0$ $\langle n \rangle_m := \langle n \rangle_m$
----	---	-----------------	---

Mit dem Befehl LC kann die Speicherzelle n gelöscht werden, während das Markenbit erhalten bleibt.

Das Markenregister kann gelöscht werden durch:

LA	M	Lösche in A	$\langle M \rangle := 0$
----	---	-------------	--------------------------

Ist bei dem Befehl LA die Spezifikation M angegeben, so wird das Markenregister auf 0 gelöscht.



TAS - Handbuch

Dez. 68

2.4.3. Abfragen der Marke

Das Markenregister kann durch zwei bestimmte Sprungbefehle abgefragt werden. In Abhängigkeit von einem entsprechenden Inhalt des Markenregisters ist es mit Hilfe eines Sprunges möglich, eine Verzweigung im Programm durchzuführen.

SM	m	Springe wenn Marke	$\langle M \rangle = L$
----	---	--------------------	-------------------------

Ist die Sprungbedingung erfüllt, wird das Markenregister auf 0 gelöscht.

SMN	m	Springe wenn Marke nicht	$\langle M \rangle = 0$
-----	---	--------------------------	-------------------------

Ist die Sprungbedingung nicht erfüllt, wird das Markenregister auf 0 gelöscht.

Beide Sprungbefehle hinterlassen in jedem Fall ein auf 0 gelöschttes Markenregister.

2.5. Typenkennung

Alle Festkommazahlen haben die Typenkennung 1. Die Ausführung der meisten Befehle hängt von der Typenkennung ab. Da eine Reihe von Befehlen nur Festkommazahlen verarbeiten kann, ist jeweils unter "Voraussetzungen" angegeben, daß die Operanden nur die Typenkennung 1 haben dürfen.

Ist diese Voraussetzung nicht erfüllt, so erfolgt die Meldung "TK-Alarm". Diese Meldung wird jedoch vom Rechenwerk vorerst nur festgehalten und führt nicht zu einer Unterbrechung.

Wenn eine falsche Typenkennung möglich ist, so wird man zweckmäßigerweise mit dem Befehl SAT eine Abfrage durchführen, ob TK-Alarm vorliegt. Im Alarmfall löscht dieser Befehl die Meldung "TK-Alarm" und springt an eine andere Stelle des Programms, um dort den TK-Alarm entsprechend abzuhandeln. Soll der TK-Alarm ignoriert werden, so kann ein Sprung auf die nächste Zeile erfolgen. Der Befehl hat dann nur die Wirkung, daß der TK-Alarm gelöscht wird.

Bleibt der TK-Alarm bestehen, so werden diejenigen Befehle, die nur das Befehlswerk benutzen, noch ausgeführt. Erst dann, wenn ein Befehl das Rechenwerk benutzen will, wird vor der Ausführung ein Eingriff mit der Fehlermeldung "TK-Alarm" gegeben. Weiteres siehe Abschnitt Fehlerbehandlung.

Sind die Bedingungen für einen TK- und einen BÜ-Alarm gegeben, dann wird der TK-Alarm stets vorrangig behandelt.

Die Typenkennung kann auch als Verzweigungskriterium verwendet werden. Es besteht in diesem Fall die Möglichkeit, in Abhängigkeit von der Typenkennung in einem bestimmten Register einen Sprung auszuführen.

ST	p s	Springe wenn Typenkennung	$\langle s_2 \rangle_t = s_1$
----	-----	---------------------------	-------------------------------

STN	p s	Springe wenn Typenkennung nicht	$\langle s_2 \rangle_t \neq s_1$
-----	-----	---------------------------------	----------------------------------

Bei erfüllter Sprungbedingung erfolgt ein relativer Sprung um p :

$$\langle F \rangle := \langle F \rangle \pm p$$

2.6. Einerrücklauf

Der Einerrücklauf tritt auf, wenn

- die Operanden verschiedene Vorzeichen haben und das Ergebnis positiv ist
 $[+] + [-] \rightarrow [+]$
 $[-] + [+] \rightarrow [+]$
- die Operanden beide negativ sind
 $[-] + [-] \rightarrow [-]$

Dabei entsteht in der nicht mehr vorhandenen 49. Stelle des Ergebniswortes (also links von der höchsten vorhandenen Stelle) ein L. Es wird zur 1. Stelle des Ergebniswortes (kleinste Stelle, also ganz rechts) hinzuaddiert. Die Zahl hat dann ihren richtigen Wert. Der Einerrücklauf tritt in der externen Darstellung nicht in Erscheinung.

Beispiele:

Die Subtraktion erfolgt in Form einer Addition. Aus diesem Grund wird intern der zweite Operand invertiert und damit negativ. Anschließend wird dann addiert. $a-b = a+(-b)$

a) $(+10) - (+3) \cong (+10) + (-3)$

oo	LOLO	$\cong +10$
LL	LL00	$\cong -3$
L	LL	OLLO
	→	L
oo	OLLL	$\cong +7$

b) $-(+10) - (+3) \cong +(-10) + (-3)$

LL	OLOL	$\cong -10$
LL	LL00	$\cong -3$
L	LL	000L
	→	L
LL	00LO	$\cong -13$

2.7. Null als Ergebnis

Innerhalb einer Operation kann die Zahl "Null" in zwei Versionen auftreten. Werden z.B. bei einer positiven Zahl alle Bits (einschließlich der Vorzeichenbits) invertiert, so wird die Zahl negativ (siehe Abschnitt 2.2.). Das gleiche geschieht auch beim Invertieren der positiven Null; man erhält die negative Null.

$$\begin{aligned} \text{oo } 0000 &= +0 \\ \text{LL } 1111 &= -0 \end{aligned}$$

Bei mathematischen Operationen wird beim Ergebnis Null im allgemeinen die negative Null erzeugt. Diese Tatsache ist jedoch ohne Bedeutung, da in jedem Fall richtig gerechnet wird. In einigen Sonderfällen muß man allerdings darauf achten, daß die Null eine negative Null sein kann.

2.8. Betrag

Der Betrag einer Festkommazahl wird so gebildet, wie es in der Mathematik üblich ist. Dabei ist der Betrag einer positiven Zahl die Zahl selbst, während bei einer negativen Zahl der Betrag dieser Zahl mit umgekehrtem Vorzeichen entspricht.

$$|+135| = +135$$

$$|-168| = +168$$

Intern werden bei der Betragsbildung alle Bits invertiert, wenn es sich um eine negative Zahl handelt. Damit ist die Zahl positiv.

2.9. Rundung

Die Rundung von Festkommazahlen erfolgt wie bei Dezimalzahlen. Ist die erste (linke) derjenigen Stellen, die durch Rundung wegfallen sollen, gleich oder größer 5, wird aufgerundet; im anderen Fall wird abgerundet.

$$365,499 = 365$$

$$365,500 = 366$$

Bei einem Teil der Befehle wird das Ergebnis automatisch gerundet, im anderen Fall kann eine Rundung mit dem Schiftbefehl SH erreicht werden.

Beim Runden einer positiven Zahl wird folgende Operation durchgeführt:

Ist von den fortfallenden Stellen die linke Stelle L, so wird die gerundete Zahl um L erhöht.

Abrunden:	Aufrunden:
$\begin{array}{r} \text{oo, LOLO} : \text{OL} \triangleq +0,640 \\ \text{oo, LOLO} \\ + \quad \text{0} \leftarrow \\ \hline \text{oo, LOLO} \triangleq +0,625 \end{array}$	$\begin{array}{r} \text{oo, LOLO} : \text{LO} \triangleq +0,65625 \\ \text{oo, LOLO} \\ + \quad \text{L} \leftarrow \\ \hline \text{oo, LOLL} \triangleq +0,6875 \end{array}$

Negative Zahl: Ist von den fortfallenden Stellen die linke Stelle 0, so wird die gerundete Zahl um L erniedrigt.

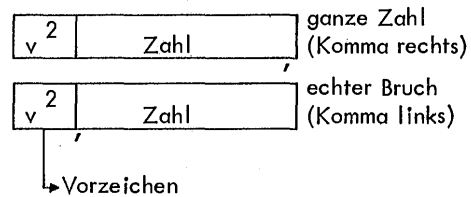
Abrunden:	Aufrunden:
$\begin{array}{r} \text{LL, OLOL} : \text{LO} \triangleq -0,640 \\ \text{LL, OLOL} \\ - \quad \text{0} \leftarrow \\ \hline \text{LL, OLOL} \triangleq -0,625 \end{array}$	$\begin{array}{r} \text{LL, OLOL} : \text{OL} \triangleq -0,65625 \\ \text{LL, OLOL} \\ - \quad \text{L} \leftarrow \\ \hline \text{LL, OL00} \triangleq -0,6875 \end{array}$

Für die Rundung wird nur die linke Vorzeichenstelle abgefragt.

2.10. Komma

2.10.1. Komma links, rechts

Festkommazahlen werden als ganze Zahlen (Komma rechts) oder als echte Brüche (Komma links) in den Speicher gebracht.



Komma links bedeutet immer, daß es zwischen den Vorzeichenstellen und der Zahl steht.

Um eine möglichst einfache Kommarechnung zu erreichen, wird das Komma für Zahlen, die das Vielfache einer Einheit sind, rechts und für Zahlen, die Teile einer Einheit sind, links angenommen. Bei doppelt langen Ergebnissen kann das Komma auch in der Mitte des doppelt langen Registers A, Q, d.h. rechts im Register A bzw. links im Register Q stehen.

2.10.2. Komma beliebig

Das Komma kann selbstverständlich an jeder beliebigen Stelle stehen (unechter Bruch). Die Kommarechnung ist dann jedoch nicht so einfach wie bei ganzen Zahlen und echten Brüchen, da sie vom Programmierer jeweils im Programm berücksichtigt werden muß. Folgende Regeln sind dabei zu beachten:

Addition, Subtraktion:

Es dürfen nur unechte Brüche addiert bzw. subtrahiert werden, die das Komma an der gleichen Stelle haben, sonst wird das Ergebnis falsch.

Multiplikation, Division:

Es dürfen unechte Brüche auch dann multipliziert bzw. dividiert werden, wenn die Kommata an verschiedenen Stellen stehen. Die Kommaverschiebung des Ergebnisses ergibt sich aus der Summe bzw. Differenz der Kommaverschiebung der Operanden.

2.10.3. Kommaverschiebung

Eine Kommaverschiebung ist mit Hilfe des Schiftbefehls SH möglich. Ein Schift nach rechts um n Stellen bedeutet eine Verschiebung des Kommas um n Stellen nach links (Division durch 2^n). Die rechten Stellen können in ein anderes Register geschiftet werden oder gehen verloren. Gegebenenfalls muß für Rundung gesorgt werden.

Ein Schift nach links um n Stellen bedeutet dementsprechend eine Kommaverschiebung um n Stellen nach rechts (Multiplikation mit 2^n). Gehen dabei links Stellen verloren, die nicht Null sind, so wird arithmetischer Alarm gegeben.

2.11. Vergleiche bei Sprüngen

Festkommazahlen einfacher Wortlänge können entweder mit dem Inhalt des Registers H oder aber mit Null verglichen werden. In beiden Fällen gibt es folgende Möglichkeiten:

<u>Springe wenn</u>	<u><H></u>	<u>±0</u>
identisch	SI	SI0
nicht identisch	SNI	SN0
größer	SG	SG0
größer gleich	SGG	SGG0
kleiner	SK	SK0
kleiner gleich	SKG	SKG0

Der Vergleich erfolgt wie bei Dezimalzahlen. Das Vorzeichen wird stets berücksichtigt; positive und negative Null sind gleichwertig.

Ist eine angegebene Bedingung erfüllt, so wird ein Sprung auf den Befehl mit der Adresse, die im Adressenteil angegeben ist, ausgeführt. Vergleiche dieser Art dienen im allgemeinen zur Programmverzweigung.

Als weitere Sprungkriterien sind möglich:

- rechtes Bit vom Register A
- Markenregister
- Typenkennung
- Merklichter
- Arithmetischer Alarm
- Typenkennungsalarm

Nähere Einzelheiten zu den Sprungbefehlen sind im Abschnitt "Sonstige Operationen" erläutert.

3. OPERATIONEN FÜR EINFACHE WORTLÄNGE

Eine genaue Trennung der Operationen nach der Wortlänge ist nicht möglich. Diejenigen Operationen, bei denen in dieser Hinsicht Zweifel auftreten, sind in den Abschnitten 4 und 5 näher beschrieben.

3.1. Bringen

Zur Verarbeitung werden die Festkommazahlen mit Hilfe der Bringebefehle in die Register A, Q, D oder H des Rechenwerkes gebracht. Dabei sind bei den Bringebefehlen, die ein Ganzwort aus dem Speicher holen, nur geradzahlige Adressen zugelassen; ungeradzahlige Adressen werden stets um 1 vermindert.

Bei Festkommazahlen (Typenkennung 1) wird mit den meisten Bringebefehlen die Marke berücksichtigt. Diese Wirkung wird bei der Beschreibung der Befehle durch die Gleichung

$$\langle x \rangle_1 := \langle x \rangle_2 \quad \langle M \rangle \vee \langle n \rangle m$$

x: Register A, Q, D oder H

zum Ausdruck gebracht und in der "Großen Befehlsliste" durch das Zeichen x in der Spalte <M>. Alle Befehle, bei denen diese Gleichung angegeben ist, setzen bei einem markierten Zahlwort (1. Bit = L) das Markenregister auf L, unabhängig davon, ob es vorher gesetzt (L) oder nicht gesetzt (0) war. Während des Transportes wird dann das erste Bit dem zweiten Bit (Vorzeichenbit) angeglichen, so daß im Register das Zahlwort 2 Vorzeichenstellen hat, die gleich sind. Im Speicher bleibt die Zahl unverändert.

Mit dem Befehl RT können die Inhalte zweier Register gegeneinander vertauscht werden.

Übersicht über die im folgenden näher erläuterten Bringebefehle:

- 3.1.1. "Bringen nach Register A"
- 3.1.2. "Bringen in andere Register"
- 3.1.3. "Bringen in Register und reservieren"
- 3.1.4. "Bringen und speichern"
- 3.1.5. "Bringen von zwei Wörtern"
- 3.1.6. "Bringen von Halbwörtern"
- 3.1.7. "Bringen von Drittelwörtern"
- 3.1.8. "Bringen eines Adressenteils"
- 3.1.9. "Sonstige Bringebefehle"

3.1.1. Bringen nach Register A

Für das Bringen einer Festkommazahl einfacher Wortlänge aus dem Kernspeicher in das Register A sind die folgenden Befehle vorgesehen:

B	n	Bringe	$\langle A \rangle := \langle n \rangle$
BN	n	Bringe negativ	$\langle A \rangle := -\langle n \rangle$
BB	n	Bringe Betrag	$\langle A \rangle := \langle n \rangle $

Ist die Festkommazahl im Speicher markiert, so wird das Markenregister gesetzt.

Mit dem Befehl BN wird die Festkommazahl aus der Speicherzelle n mit umgekehrtem Vorzeichen (d.h. invertiert) ins Register A gebracht. Der Befehl BB veranlaßt den Transport des Betrages einer Festkommazahl aus dem Speicher ins Register A. Ist die Festkommazahl positiv, so ist ihr Betrag identisch; ist sie negativ, so wird das Vorzeichen umgekehrt. Damit ist die Zahl positiv.

Die übrigen Rechenwerksregister werden durch diese Befehle nicht verändert.

Die Festkommazahlen im Speicher sind nie über- oder untergelaufen.

3.1.2. Bringen in andere Register

Außer in das Register A kann eine Festkommazahl einfacher Wortlänge aus dem Speicher in eines der übrigen Rechenwerksregister Q, D oder H gebracht werden. Dazu stehen folgende Befehle zur Verfügung:

BQ	n	Bringe nach Q	$\langle Q \rangle := \langle n \rangle$
BD	n	Bringe nach D	$\langle D \rangle := \langle n \rangle$
BH	n	Bringe nach H	$\langle H \rangle := \langle n \rangle$
BQB	n	Bringe nach Q und Bringe (nach A)	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n \rangle$

Mit dem Befehl BQB wird eine Festkommazahl aus der Speicherzelle n in das Register Q und außerdem gleichzeitig in das Register A gebracht.

Ist die Festkommazahl im Speicher markiert, so wird das Markenregister gesetzt. Im Speicher ist eine Festkommazahl nie über- oder untergelaufen.

3.1.3. Bringen in Register und reservieren

Bisweilen ist es nötig, daß vor dem Transport aus dem Speicher in das Register A der alte Registerinhalt für spätere Operationen reserviert wird. Diese Möglichkeit ist mit den Befehlen

BR	n	Bringe und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := \langle n \rangle$
BNR	n	Bringe negativ und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := -\langle n \rangle$

gegeben. Erst dann, wenn der alte Inhalt des Registers A im Register H sichergestellt ist, wird die Festkommazahl einfacher Wortlänge aus der Speicherzelle n in das Register A gebracht. (Mit dem Befehl CR kann die Reserve wieder zurückgeholt werden.)

H

TAS - Handbuch

Dez. 68

Bei dem Befehl BNR wird die Zahl mit umgekehrtem Vorzeichen (d.h. invertiert) aus dem Speicher ins Register A gebracht.
 War die Festkommazahl im Speicher markiert, so wird das Markenregister gesetzt. Die Zahl im Speicher ist nie über- oder untergelaufen.

3.1.4. Bringen und speichern

Um den Inhalt des Registers A mit dem Inhalt der Speicherzelle n zu vertauschen, steht der Befehl

BC	n	Bringe und speichere	$\langle A \rangle := \langle n \rangle$
----	---	----------------------	--

zur Verfügung.

War der Inhalt des Registers A und der Speicherzelle n ein Zahlwort einfacher Wortlänge, so bleibt die ursprüngliche Markenstelle in der Speicherzelle erhalten.

War die Festkommazahl im Speicher markiert, so wird das Markenregister gesetzt. Die Zahlen im Register und im Speicher dürfen nicht über- oder untergelaufen sein.

3.1.5. Bringen von zwei Wörtern

Zwei Festkommazahlen einfacher Wortlänge, die in hintereinanderliegenden Speicherzellen stehen, können mit den Befehlen

BZ	n	Bringe zwei Wörter	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n+2 \rangle$
BZN	n	Bringe zwei Wörter negativ	$\langle A \rangle := -\langle n \rangle$ $\langle Q \rangle := -\langle n+2 \rangle$

in die Register A und Q gebracht werden. Die Zahlen können dabei verschiedene Vorzeichen haben. Bei dem Befehl BZ stehen nach dem Transport aus dem Speicher die Zahlen mit ihren Vorzeichen in den Registern A und Q; bei BZN erscheinen sie dort mit umgekehrten Vorzeichen (d.h. invertiert). Sind beide oder eine der zwei Zahlen markiert, so wird das Markenregister auf L gesetzt.

Der Befehl BZ hat dieselbe Wirkung wie die zwei aufeinanderfolgenden Befehle B n, BQ n + 2.

Mit den Befehlen BZ und BZN können auch Zahlen doppelter Wortlänge in das doppelt lange Register A, Q gebracht werden. (näheres siehe unter Abschnitt 4.)

3.1.6. Bringen von Halbwörtern

Um den Speicherbereich besser auszunutzen, können kleinere Festkommazahlen in Halbwörtern untergebracht werden. Wird eine Festkommazahl halber Wortlänge in das Register A gebracht, so besteht die Möglichkeit, sie wie eine Zahl einfacher Wortlänge zu behandeln. Die Zahlen können mit und ohne Vorzeichen verarbeitet werden.

Eine Festkommazahl halber Wortlänge mit Vorzeichen ist eine ganze Zahl und liegt im Bereich von $0 \dots \pm(2^{23} - 1)$.

Für das Bringen der Zahlen ins Register A sind die Befehle

B2V	m	Bringe Halbwort mit Vorzeichen	$\langle A \rangle := \langle m \rangle$
B2VN	m	Bringe Halbwort mit Vorzeichen negativ	$\langle A \rangle := -\langle m \rangle$

vorgesehen. Bei dem Befehl B2V steht die Zahl nach dem Transport aus dem Speicher mit ihrem Vorzeichen im Register A, beim Befehl B2VN wird sie mit umgekehrtem Vorzeichen (d.h. invertiert) dorthin gebracht. Die Typenkennung im Register A ist bei diesen beiden Befehlen 1.

Eine Festkommazahl halber Wortlänge ohne Vorzeichen ist eine ganze Zahl, stets positiv und liegt im Bereich von $0 \dots +(2^{24} - 1)$. Sie wird mit dem Befehl

B2	m	Bringe Halbwort	$\langle A \rangle := \langle m \rangle$
----	---	-----------------	--

ins Register A gebracht.

Die Festkommazahlen halber Wortlänge sind im allgemeinen ganze Zahlen (Kommastellung rechts); es können aber auch echte Brüche dargestellt werden. Nach einem Linksschift (SH AL 24) steht dann im Register A die Festkommazahl als echter Bruch (Komma links) und kann wie eine Festkommazahl einfacher Wortlänge weiterverarbeitet werden.

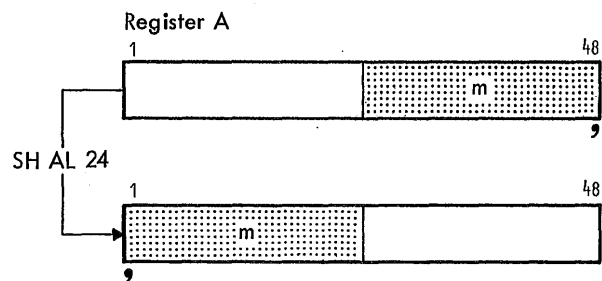


Bild 3.1 Umwandlung einer ganzen Festkommazahl halber Wortlänge in einen echten Bruch

3.1.7. Bringen von Drittelwörtern

Eine Festkommazahl kann auch die Länge eines Drittelwortes (16 Binärstellen) haben. Nach ihrem Transport vom Speicher in das Register A kann sie bei der Weiterverarbeitung wie eine Zahl einfacher Wortlänge behandelt werden. Eine Marke ist nicht vorhanden. Die Zahlen werden mit oder ohne Vorzeichen verarbeitet. Festkommazahlen von der Länge eines Drittelwortes mit Vorzeichen sind ganze Zahlen im Bereich von $0 \dots \pm(2^{16} - 1)$ (das erste Bit ist Vorzeichenstelle). Sie werden durch den Befehl

B3V	m	Bringe Drittelwort mit Vorzeichen	$\langle A \rangle := \langle m \rangle_{9-24}$
-----	---	-----------------------------------	---

mit Vorzeichen aus dem Speicher ins Register A gebracht.

Festkommazahlen von der Länge eines Drittelwortes ohne Vorzeichen sind ganze Zahlen, stets positiv und liegen im Bereich von $0 \dots + (2^{16} - 1)$. Sie werden mit dem Befehl

B3	m	Bringe Drittelwort	$\langle A \rangle := \langle m \rangle_{0-24}$
----	---	--------------------	---

in das Register A gebracht.

Die linken 8 Bits können zusätzlich für eine weitere, dazugehörige Information ausgenutzt werden. Die Typenkennung wird bei beiden Befehlen auf 1 gesetzt.

3.1.8. Bringen eines Adressenteils

Eine Festkommazahl kann auch eine Konstante sein. Sie steht im Adressenteil der Befehle

BA	z	Bringe Adressenteil	$\langle A \rangle := z$
BAN	z	Bringe Adressenteil negativ	$\langle A \rangle := -z$

und hat den Wert einer ganzen Zahl von $0 \dots 65\,535$.

Durch Modifizierung kann dieser Wert verändert werden und ein Vorzeichen (eine Binärstelle) erhalten. Er liegt dann im Bereich von $0 \dots \pm (2^{23} - 1)$ und steht beim Befehl BA nach dem Transport mit seinem Vorzeichen, bei BAN mit umgekehrtem Vorzeichen (d.h. invertiert) im Register A.

Wenn es erforderlich ist, kann man mit den Befehlen

BAR	z	Bringe Adressenteil und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := z$
BANR	z	Bringe Adressenteil negativ und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := -z$

zunächst den alten Inhalt des Registers A im Register H sicherstellen. Danach erfolgt der Transport der Konstanten genau wie bei den Befehlen BA und BAN. Die Typenkennung im Register A wird bei diesen Befehlen auf 1 gesetzt.

Mehrere Festkommazahlen, die im Speicher in hintereinanderliegenden Zellen stehen, können mit den Befehlen WTV und WTR in ebenfalls hintereinanderliegenden Speicherzellen an eine andere Stelle im Speicher gebracht werden.

Mit einigen Tabellensuchbefehlen kann aus einer Tabelle von Festkommazahlen die gesuchte Zahl herausgefunden werden.

3.1.9. Sonstige Bringebefehle

Um den Inhalt einer Speicherzelle unabhängig von der Typenkennung in das Register A zu bringen, steht der Befehl

BU	n	Bringe unverändert	$\langle A \rangle := \langle n \rangle$
----	---	--------------------	--

zur Verfügung. Es wird ein Bitmuster gebracht, d.h. alle 48 Informationsbits werden unverändert in das Register A transportiert. (Das in 2.4. über die Marke Geschriebene bleibt unberücksichtigt.)

Mit dem Befehl BU ist es möglich, über- oder untergelaufene Zahlwörter, die zuvor mit dem Befehl CU abgespeichert wurden, in das Register A zu bringen.

Zwei hintereinanderliegende Halbwörter werden mit dem Befehl

BZ2	m	Bringe zwei Halbwörter	$\langle A \rangle := \langle m, m+1 \rangle$
-----	---	------------------------	---

in das Register A gebracht, wobei die Typenkennung des Ganzwortes übernommen wird, in dem das erste Halbwort steht. Ist die Adresse m geradzahlig, so wirkt dieser Befehl wie BU. Ein Unterschied zwischen beiden Befehlen besteht in diesem Fall darin, daß der Befehl BZ2 als Zweitcode bei dem Befehl R nicht zugelassen ist. Mit BZ2 wird wie bei BU ein Bitmuster in das Register A gebracht.

3.2. Speichern

Die Ergebnisse werden durch die Speicherbefehle in den Speicher zurückgebracht. Festkommazahlen dürfen dabei den vorgesehenen Zahlenbereich nicht überschreiten, d.h. sie dürfen nicht über- oder untergelaufen sein. Intern gesehen müssen das 1. und das 2. Bit gleich sein. Im anderen Fall wird Bereichsüberschreitungsalarm gegeben (BÜ-Alarm), und das Zahlwort wird nicht abgespeichert. Eine Ausnahme bildet der Befehl CU.

Zahlen sind im Speicher nicht markiert. Das 1. Bit in der Speicherzelle wird im allgemeinen 0 gesetzt. Eine Ausnahme stellen die Befehle CMT, CMR und CMC dar (siehe auch Abschnitt 2.4.). Durch ihre Anwendung kann im Speicher eine Zahl markiert werden.

CMT das Zahlwort wird markiert $\langle n \rangle_n := L$

CMR der Inhalt des Registers M wird abgespeichert $\langle n \rangle_n := \langle M \rangle$

CMC die Markenstelle im Speicher bleibt erhalten $\langle n \rangle_n := \langle n \rangle_n$

In allen anderen Fällen ist das abgespeicherte Zahlwort nicht markiert.

Übersicht über die im folgenden näher erläuterten Speicherbefehle:

- 3.2.1. "Speichern aus Register A"
- 3.2.2. "Speichern aus anderen Registern"
- 3.2.3. "Speichern aus Register A und reservieren"
- 3.2.4. "Speichern und bringen"
- 3.2.5. "Speichern von zwei Wörtern"
- 3.2.6. "Speichern von Halbwörtern"
- 3.2.7. "Speichern von Drittelwörtern"
- 3.2.8. "Speichern mit Marke"
- 3.2.9. "Sonstige Speicherbefehle"



TAS - Handbuch

Juli 69

3.2.1. Speichern aus Register A

Eine Festkommazahl einfacher Wortlänge, die im Register A steht, kann mit den Befehlen

C	n	Speichere	$\langle n \rangle := \langle A \rangle$
CN	n	Speichere negativ	$\langle n \rangle := -\langle A \rangle$
CB	n	Speichere Betrag	$\langle n \rangle := \langle A \rangle $

in den Speicher zurückgebracht werden.

Der Befehl CN speichert die Zahl mit ihrem umgekehrten Vorzeichen (d.h. invertiert) ab. Mit dem Befehl CB wird der Betrag der Festkommazahl aus dem Register A zurückgespeichert. Ist die Zahl positiv, so ist ihr Betrag identisch mit der Zahl; ist sie negativ, so wird die Zahl vor dem Abspeichern invertiert; damit ist die Zahl positiv (siehe Abschnitt 2.8.).

Bei Anwendung der Befehle C, CN und CB sind die Zahlen im Speicher nicht markiert. Bei Über- oder untergelaufenen Zahlwörtern erfolgt BÜ-Alarm.

3.2.2. Speichern aus anderen Registern

Für die Abspeicherung einer Festkommazahl einfacher Wortlänge aus einem der Register Q, D oder H stehen die Befehle

CQ	n	Speichere aus Q	$\langle n \rangle := \langle Q \rangle$
CD	n	Speichere aus D	$\langle n \rangle := \langle D \rangle$
CH	n	Speichere aus H	$\langle n \rangle := \langle H \rangle$

zur Verfügung.

Bei Anwendung dieser Befehle sind die Zahlen im Speicher nicht markiert. Bei Über- oder untergelaufenen Zahlwörtern erfolgt BÜ-Alarm.

3.2.3. Speichern aus Register A und Reserve bringen

Mit dem Befehl

CR	n	Speichere und bringe Reserve	$\langle n \rangle := \langle A \rangle$ $\langle A \rangle := \langle H \rangle$
----	---	------------------------------	--

kann eine Festkommazahl einfacher Wortlänge aus dem Register A in den Speicher zurückgebracht werden. Außerdem wird anschließend der Inhalt des Registers H in das Register A gebracht. Der Befehl CR kann z.B. im Zusammenhang mit dem Befehl BR bzw. BNR verwendet werden. Bei Anwendung des Befehls CR sind die Zahlen im Speicher nicht markiert. Bei Über- oder untergelaufenen Zahlwörtern im Register A erfolgt BÜ-Alarm.

3.2.4. Speichern und bringen

Das Vertauschen zweier Festkommazahlen einfacher Wortlänge, die im Register A und in der Speicherzelle n stehen, ist mit dem Befehl

BC	n	Bringe und Speichere	$\langle n \rangle := \langle A \rangle$
----	---	----------------------	--

möglich. War die Zahl im Speicher markiert, so bleibt das Markenbit dort erhalten und das Markenregister wird gesetzt.

Die Ausführung ist die gleiche, wie wenn die Befehle B n und CMC n gleichzeitig angewendet würden.

3.2.5. Speichern von zwei Wörtern

Mit Hilfe des Befehls

CZ	n	Speichere zwei Wörter	$\langle n \rangle := \langle A \rangle$ $\langle n+2 \rangle := \langle Q \rangle$
----	---	-----------------------	--

können zwei Festkommazahlen einfacher Wortlänge aus den Registern A und Q in zwei hintereinanderliegende Speicherzellen zurückgebracht werden. Dieser Befehl hat die gleiche Wirkung wie die beiden aufeinanderfolgenden Befehle C n, CQ n+2.

Mit CZ ist es außerdem möglich, ein doppelt langes Ergebnis aus dem doppelt langen Register AQ in den Speicher zurückzubringen (siehe Abschnitt 4.).

Bei Anwendung des Befehls sind die Zahlen im Speicher nicht markiert. Bei Über- oder untergelaufenen Zahlwörtern erfolgt BÜ-Alarm.

3.2.6. Speichern von Halbwörtern

Für das Abspeichern kleinerer ganzer Festkommazahlen, die in einem Halbwort untergebracht werden können, steht der Befehl

C2	m	Speichere Halbwort	$\langle m \rangle := \langle A \rangle_{25-48}$
----	---	--------------------	--

zur Verfügung. Es werden aus dem Register A sowohl Zahlen mit Vorzeichen (Bereich von $0 \dots \pm(2^{23}-1)$), als auch ohne Vorzeichen (Bereich von $0 \dots +(2^{24}-1)$) in die Speicherzelle m gebracht.

Die andere Hälfte des zugehörigen Ganzwortes im Speicher und die Typenkennung bleiben unverändert. Die Halbwörter sind dort nicht markiert.

Es können auch echte Brüche in Halbwörtern abgespeichert werden. In diesem Fall steht die Zahl linksbündig im Register A. Da nun das Komma ebenfalls links, d.h. hinter der Vorzeichenstelle angenommen wird, muß zuerst ein Rechtsschift (SH A 24) um 24 Bits, ggf. mit Rundung (SH AR 24), durchgeführt werden. Damit steht dann das abzuspeichernde Halbwort in den Stellen 25 - 48 des Registers A.

3.2.7. Speichern von Drittelwörtern

Kleinere ganze Zahlen, die den Wert von $0 \dots (2^{18}-1)$ nicht überschreiten und im Register A rechtsbündig stehen, werden mit dem Befehl

C3	m	Speichere Drittelwort	$\langle m \rangle_{9-24} := \langle A \rangle_{33-48}$
----	---	-----------------------	---

in das adressierte Halbwort im Speicher zurückgebracht. Die restlichen linken 8 Bits und das zugehörige Markenbit bleiben unverändert.

3.2.8. Speichern mit Marke

Bei den Befehlen

CMT	n	Speichere <u>markiert</u>	$\langle n \rangle := \langle A \rangle$ $\langle n \rangle_m := L$
CMR	n	Speichere mit <u>Marke</u> aus <u>Register</u>	$\langle n \rangle := \langle A \rangle$ $\langle n \rangle_m := \langle M \rangle$
CMC	n	Speichere mit <u>Marke</u> aus <u>Speicher</u>	$\langle n \rangle := \langle A \rangle$ $\langle n \rangle_m := \langle n \rangle_m$

wird eine Marke berücksichtigt.

Um ein Zahlwort, das vom Register A in den Speicher gebracht worden ist, auf jeden Fall zu markieren, wird der Befehl CMT angewendet. Das erste Bit in der Speicherzelle n wird durch CMT immer auf L gesetzt.

Der Befehl CMR bewirkt, daß die Markenstelle eines Zahlwortes, das aus dem Register A abgespeichert wurde, mit dem Inhalt des Registers M belegt wird. Die Zahl im Speicher ist also dann markiert, wenn im Register M ein L steht.

Der Befehl CMC ermöglicht die Abspeicherung einer Zahl aus dem Register A, wobei das Markenbit, das in der Speicherzelle n steht, unverändert bleibt. Während also die im Speicher vorhandene Zahl durch die neue Zahl aus dem Register A überschrieben wird, bleibt die ursprüngliche Markenstelle erhalten.

3.2.9. Sonstige Speicherbefehle

Um ein Bitmuster von der Länge eines Ganzwortes aus dem Register A abzuspeichern, steht der Befehl

CU	n	Speichere <u>unverändert</u>	$\langle n \rangle := \langle A \rangle$
----	---	------------------------------	--

zur Verfügung. In die adressierte Speicherzelle werden alle 48 Informationsbits und die Typenkennung unverändert gebracht.

Mit diesem Befehl ist die Abspeicherung eines Über- oder untergelaufenen Zahlwortes ohne BÜ-Alarm möglich.

Durch den Befehl BU kann dann eine Weiterverarbeitung der Zahl erfolgen, ohne daß sie falsch wird.

Wird mit einem anderen Befehl als BU weitergearbeitet, dann ist zu beachten, daß beim Abspeichern die Vorzeichenstelle im Register (1. Bit) im Speicher zur Markenstelle, die Überlaufstelle im Register (2. Bit) im Speicher zur Vorzeichenstelle geworden ist. Das kann bei der weiteren Verarbeitung einer über- oder untergelaufenen Zahl zu Fehlern führen.

3.3. Addition - Subtraktion

Zur Durchführung einer Subtraktion wird beim Subtrahend das Vorzeichen gewechselt (d.h. er wird invertiert). Anschließend erfolgt eine Addition der beiden Operanden. Aus diesem Grund können die Beschreibungen der Addition und der Subtraktion zusammengefaßt werden.

Um eine Addition bzw. eine Subtraktion durchführen zu können, muß der eine Operand bereits in einem Register des Befehlswerkes vorhanden sein. Der andere Operand wird durch den Additions- bzw. Subtraktionsbefehl vom Speicher in das Rechenwerk gebracht. Bezüglich der Marke gilt das im Abschnitt 3.1. über den Bringebefehl Gesagte. Mit den Befehlen AC und SBC wird das Ergebnis wieder im Speicher abgespeichert. Hier gilt sinngemäß das, was im Abschnitt 3.2. über die Speicherbefehle näher beschrieben ist.

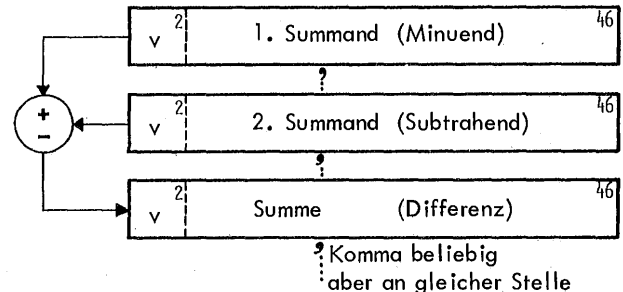


Bild 3.2 Addition (Subtr.) von Zahlen einf. Wortlänge

Ein evtl. Komma kann bei den Zahlen an beliebiger Stelle angeordnet sein, jedoch muß es bei beiden Zahlen an der gleichen Stelle stehen, sonst wird das Ergebnis falsch.

Additionsbefehle:

A	n	<u>Addiere</u>	$\langle A \rangle := \langle A \rangle + \langle n \rangle$
AC	n	<u>Addiere im Speicher</u>	$\langle n \rangle := \langle n \rangle + \langle A \rangle$
AB	n	<u>Addiere Betrag</u>	$\langle A \rangle := \langle A \rangle + \langle n \rangle $

Subtraktionsbefehle:

SB	n	<u>Subtrahiere</u>	$\langle A \rangle := \langle A \rangle - \langle n \rangle$
SBI	n	<u>Subtrahiere invers</u>	$\langle A \rangle := \langle n \rangle - \langle A \rangle$
SBC	n	<u>Subtrahiere im Speicher</u>	$\langle n \rangle := \langle n \rangle - \langle A \rangle$
SBB	n	<u>Subtrahiere Betrag</u>	$\langle A \rangle := \langle A \rangle - \langle n \rangle $
SBD	n	<u>Subtrahiere von D</u>	$\langle A \rangle := \langle D \rangle - \langle n \rangle$

Bei den vorstehenden Befehlen muß der eine Operand im Register A (Register D bei SBD) sein. Er sollte nicht über- oder untergelaufen sein und die Typenkennung 1 haben. Der andere Operand wird aus der Speicherzelle mit der Adresse n geholt. Er soll ebenfalls die Typenkennung 1 haben. Dabei wird auch die Markenstelle berücksichtigt. Das Ergebnis steht dann im Register A (im Speicher bei AC bzw. SBC) mit der Typenkennung 1. Ist das Ergebnis über- oder untergelaufen, wird BÜ-Alarm gegeben.

Ist der Operand im Register über- oder untergelaufen, so wird eine Addition oder Subtraktion trotzdem ausgeführt. In diesem Fall ist es jedoch möglich, daß das Ergebnis so weit überläuft, daß es falsch wird.

Beispiel:

$$\begin{array}{r}
 01, L \dots \hat{=} +1,5 \dots \\
 + 00, L \dots \hat{=} +0,5 \dots \\
 \hline
 10, 0 \dots \hat{=} -1,5 \dots
 \end{array}$$



Im vorstehenden Beispiel ist im Ergebnis nicht nur das Überlaufbit (2. Bit) ungleich dem Vorzeichenbit (1. Bit), sondern auch das Vorzeichenbit verändert. Die Ungleichheit der ersten beiden Bits führt zu BÜ-Alarm. Die Änderung des Vorzeichens wird jedoch nicht erkannt, so daß das Ergebnis nicht nur übergelaufen, sondern auch falsch ist.

Der aus dem Speicher geholte Operand wird vom Rechner als nicht über- oder untergelaufen betrachtet, auch wenn er zuvor z.B. mit dem Befehl CU als über- bzw. untergelaufene Zahl abgespeichert wurde.

AA	z	Addiere Adressenteil	$\langle A \rangle := \langle A \rangle + z$
SBA	z	Subtrahiere Adressenteil	$\langle A \rangle := \langle A \rangle - z$

Bei den Befehlen AA und SBA wird zum Inhalt des Registers A eine Konstante addiert, die im Adressenteil des Befehls steht. Sie kann den Wert von maximal 65 535 (Komma rechts angenommen, $2^{16}-1$) annehmen. Durch Modifizierung kann dieser Wert verändert werden und dann maximal $\pm(2^{23}-1)$ werden.

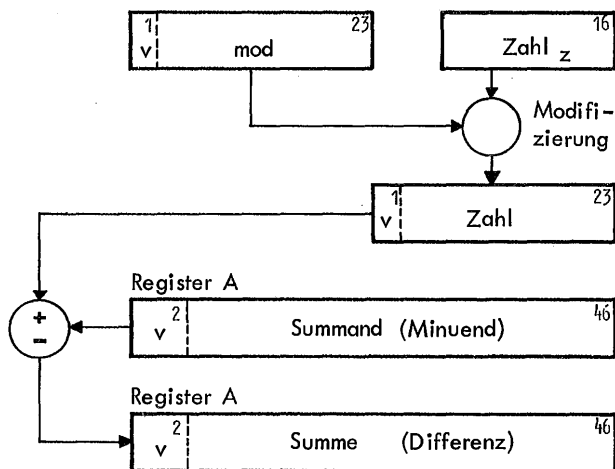


Bild 3.3 Beispiel zu den Befehlen AA und SBA

Während bei den anderen Befehlen - unabhängig von der Typenkennung - immer nach Festkommaart gerechnet wird, ist die Wirkung bei den Befehlen AA und SBA abhängig von der Typenkennung des Operanden im Register. Das Ergebnis erhält die höhere der Typenkennungen der beiden Operanden.

3.4. Multiplikation

Bei der Multiplikation von Festkommazahlen einfacher Wortlänge hat das Produkt doppelte Wortlänge. Das Ergebnis der Multiplikation kann mit doppelter Wortlänge weiter benutzt oder aber auf einfache Wortlänge gerundet werden.

3.4.1. Ergebnis mit doppelter Wortlänge

Für ein doppelt langes Ergebnis sind folgende Befehle vorhanden:

ML	n	Multipliziere	$\langle A, Q \rangle := \langle A \rangle \cdot \langle n \rangle$
MLN	n	Multipliziere negativ	$\langle A, Q \rangle := -\langle A \rangle \cdot \langle n \rangle$

Soll zusätzlich ein doppelt langes Wort hinzuaddiert werden, so gelten die Befehle

MLA	n	Multipliziere akkumulierend	$\langle A, Q \rangle := \langle A \rangle \cdot \langle n \rangle + \langle H, Q \rangle$
MAN	n	Multipliziere akkumulierend negativ	$\langle A, Q \rangle := -\langle A \rangle \cdot \langle n \rangle + \langle H, Q \rangle$

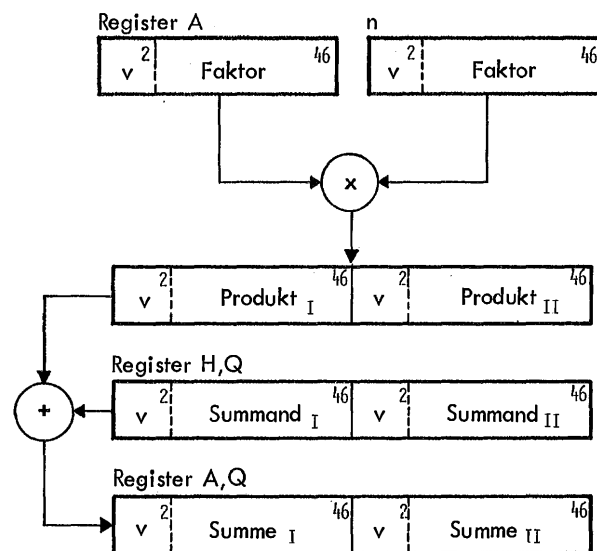


Bild 3.4 Beispiel zu den Befehlen MLA und MAN

Die beiden Faktoren stehen im Register A und in der Speicherzelle n, der Summand im doppelt langen Register H, Q. Die Typenkennung der Faktoren muß 1 sein; die Typenkennung des Ergebnisses ist stets 1.

Die Faktoren dürfen nicht über- oder untergelaufen sein, anderenfalls wird BÜ-Alarm gegeben, und das Ergebnis ist falsch. Wenn das Ergebnis durch die Addition über- oder unterläuft, wird ebenfalls BÜ-Alarm gegeben.

Für die einfache Kommarechnung ergibt sich folgende Übersicht über die möglichen Kommastellungen:

Faktor	Faktor	Ergebnis
rechts	rechts	rechts
links	links	links
rechts	links	Mitte
links	rechts	Mitte

Das Ergebnis (Produkt) der Multiplikation ist der Summand 1 der anschließenden Addition. Beim Summand 2 ($\langle H, Q \rangle$) muß das Komma an der gleichen Stelle stehen, wie beim Summand 1. Weitere Einzelheiten über doppelt lange Festkommazahlen können dem Abschnitt 4 entnommen werden.

Die Befehle sind dazu geeignet, mit Zwischenergebnissen höherer Genauigkeit zu rechnen.

3.4.2. Ergebnis mit einfacher Wortlänge

Das Produkt hat – wie vorstehend beschrieben – ebenfalls eine doppelte Wortlänge. Es wird jedoch durch Rundung auf einfache Wortlänge gekürzt. Die rechte Hälfte des Ergebnisses entfällt. Es stehen folgende Befehle zur Verfügung:

MLR	n	Multipliziere mit <u>R</u> undung	$\langle A \rangle := \langle A \rangle \cdot \langle n \rangle$
MNR	n	Multipliziere <u>n</u> egativ mit <u>R</u> undung	$\langle A \rangle := -\langle A \rangle \cdot \langle n \rangle$

Soll zusätzlich eine Festkommazahl einfacher Wortlänge hinzuaddiert werden, so gelten die Befehle

MAR	n	Multipliziere <u>a</u> kkumulierend mit <u>R</u> undung	$\langle A \rangle := \langle A \rangle \cdot \langle n \rangle + \langle H \rangle$
MANR	n	Multipliziere <u>a</u> kkumulierend <u>n</u> egativ mit <u>R</u> undung	$\langle A \rangle := -\langle A \rangle \cdot \langle n \rangle + \langle H \rangle$

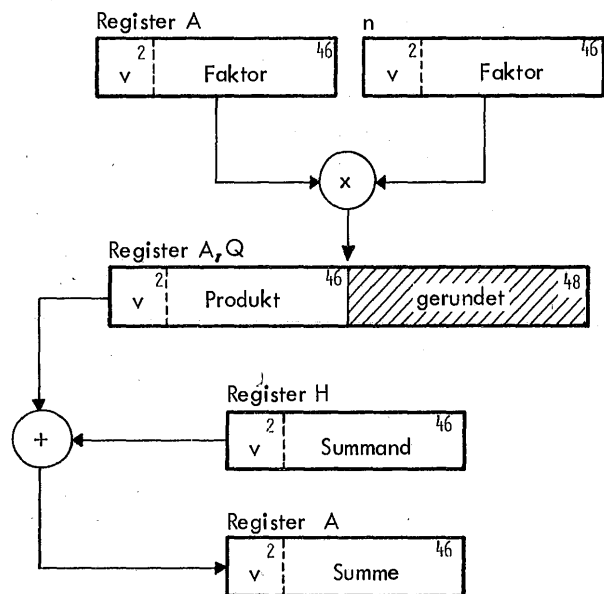


Bild 3.5 Beispiel zu den Befehlen MAR und MANR

Die beiden Faktoren stehen im Register A und in der Speicherzelle n, der Summand im Register H. Die Typenkennung der Faktoren muß 1 sein. Die Typenkennung des Ergebnisses ist stets 1.

Die Faktoren dürfen nicht über- oder untergelaufen sein. Anderenfalls wird BÜ-Alarm gegeben, und das Ergebnis ist falsch. Wenn das Ergebnis durch die Addition über- oder unterläuft, wird ebenfalls BÜ-Alarm gegeben.

Für die einfache Kommarechnung ergibt sich folgende Übersicht über die möglichen Kommastellungen:

Faktor	Faktor	Ergebnis
links	links	links
links	rechts	rechts
rechts	links	rechts

Als Beispiel sei hier 10% von 3,12 DM gerechnet

$$\begin{array}{r} 312, \times, 100 \\ \hline 031,200 \\ 031, \end{array}$$

Das Ergebnis beträgt gerundet 31 Pfennige = 0,31 DM

Für die Addition gilt, daß das Komma bei beiden Summanden beliebig angeordnet sein kann aber an gleicher Stelle stehen muß (siehe auch Abschnitt 3.3.).

3.4.3. Kommarechnung

Bei dieser Betrachtung soll das Komma des einen Faktors um x Stellen (ausgehend vom "Komma links") nach rechts versetzt sein und beim anderen Faktor um y Stellen. Für die Verschiebung des Kommas beim Ergebnis um z Stellen gilt:

$$z = x + y$$

Hierbei kann auch mit negativen Zahlen operiert werden. In diesem Fall bedeutet das eine Verschiebung des Kommas nach links. Das Komma kann auch außerhalb des Registers stehen. Drückt man die Kommaverschiebung in 2er-Potenzen aus, so gilt für die Multiplikation allgemein:

$$\begin{array}{l} \text{Produkt} = (a_1 \cdot 2^{-x}) \cdot (a_2 \cdot 2^{-y}) \\ \quad \quad = a_1 \cdot a_2 \cdot 2^{-x-y} \end{array} \left. \vphantom{\begin{array}{l} \text{Produkt} \\ \text{Produkt} \end{array}} \right\} a = \text{echte Brüche}$$

$$\begin{array}{l} \text{Produkt} = (a_1 \cdot 2^x) \cdot (a_2 \cdot 2^y) \\ \quad \quad = a_1 \cdot a_2 \cdot 2^{x+y} \end{array} \left. \vphantom{\begin{array}{l} \text{Produkt} \\ \text{Produkt} \end{array}} \right\} a = \text{ganze Zahlen}$$

Das Produkt $a_1 a_2$ wird durch den Befehl gebildet, die Kommaverschiebung (2er-Potenzen) muß vom Programmierer berücksichtigt werden.

3.5. Division

Die Anwendung der zur Verfügung stehenden Festkomma-Divisionsbefehle DV und DVI sowie DVD richtet sich bei der Verarbeitung von Festkommazahlen einfacher Wortlänge danach, ob das zu erwartende Ergebnis kleiner oder größer/gleich 1 ausfällt, d.h. mit anderen Worten, ob der Betrag des Dividenden kleiner oder größer/gleich dem Betrag des Divisors ist.

Die Ausführungen unter 3.5.1. gelten dann, wenn mit Sicherheit feststeht, daß das Ergebnis kleiner als 1 ausfällt (Anwendung der Befehle DV bzw. DVI). Im Abschnitt 3.5.2. ist als Beispiel der Fall dargestellt, daß das Ergebnis mit Sicherheit größer als 1 erwartet wird (Anwendung des Befehls DVD), während das Beispiel im Abschnitt 3.5.3. dann gilt, wenn nicht mit Bestimmtheit feststeht, ob das Ergebnis größer oder kleiner als 1 wird (Befehl DVD). In den nachfolgenden Betrachtungen wird das Komma links (hinter dem Vorzeichen) angenommen.

3.5.1. Divisions-Ergebnis kleiner 1

Für die Division stehen die Befehle

DV	n	Dividiere	$\langle A \rangle := \langle A \rangle : \langle n \rangle$
DVI	n	Dividiere invers	$\langle A \rangle := \langle n \rangle : \langle A \rangle$

zur Verfügung.

Voraussetzungen:

1. Der Divisor darf nicht Null sein;
2. Der Betrag des Dividenden muß kleiner sein als der Betrag des Divisors (bei Komma an der gleichen Stelle), mit anderen Worten: der Quotient muß kleiner als 1 sein.
Im anderen Fall ist das Ergebnis eine Gleitkommazahl; es erfolgt BÜ-Alarm.

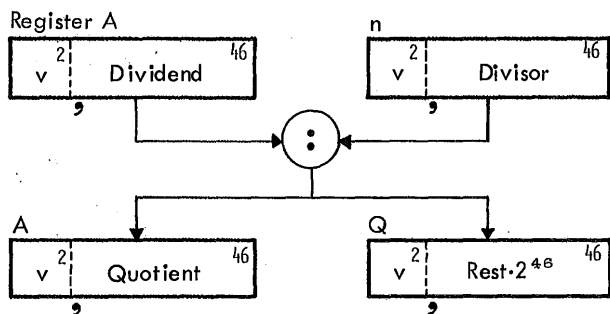


Bild 3.6 Beispiel zu den Befehlen DV und DVI

Der Quotient steht im Register A, der Rest im Register Q. Für Dividend, Divisor, Quotient und Rest wurde die gleiche Kommastellung (links, hinter dem Vorzeichen) angenommen. In diesem Fall ist der Rest mit dem Faktor 2^{46} multipliziert.

Dividend und Divisor müssen die Typenkennung 1 haben. Außerdem dürfen beide weder über- noch untergelaufen sein.

Für die einfache Kommarechnung ergibt sich folgende Übersicht über die möglichen Kommastellungen:

Dividend	Divisor	Quotient	Rest $\cdot 2^{46}$
links	links	links	links
rechts	rechts	links	rechts
rechts	links	rechts	rechts

Wenn der Betrag des Dividenden gleich oder größer als der Divisor ist (bei Komma an der gleichen Stelle), wird das Ergebnis eine Gleitkommazahl, die normalisiert, aber nicht gerundet ist. Das Ergebnis ist nur dann stellenrichtig, wenn Dividend und Divisor das Komma links haben. Es wird BÜ-Alarm gegeben.

Für die allgemeine Kommarechnung gilt, ausgehend vom Komma links und eine Kommaverschiebung

um x Stellen beim Dividend und
um y Stellen beim Divisor

eine Kommaverschiebung von

x-y Stellen für den Quotienten
x-46 Stellen für den Rest

Bei positiven Zahlen erfolgt die Kommaverschiebung nach rechts, bei negativen Zahlen nach links. Für den Gleitkommafall muß der Gleitkommaquotient mit dem Faktor 2^{x-y} multipliziert werden. Die Umwandlung einer Gleitkommazahl in eine Festkommazahl ist im Abschnitt 3.7. beschrieben.

$$060 : 070 = 857 \text{ Rest } 010$$

Register		n		A		Q	
A	Q	Divisor	=	Quotient	=	Rest	
Dividend	0	Divisor	=	Quotient	=	Rest	
060	000	070	=	857	=	010	
060	000						
056	000						
004	000						
003	500						
000	500						
000	490						
000	010						
Komma links							
,060		,070	=	,857	=	,000 010	
Komma rechts							
060 ,		070 ,	=	,857	=	,010	
Komma gemischt							
060 ,		,070	=	857 ,	=	,010	
Komma gemischt							
,060		070 ,	=	,000 857	=	,000 010	

Bild 3.7 Dezimales Beispiel einer Division mit dem Befehl DV bei Dividend kleiner als Divisor

3.5.2. Divisions-Ergebnis größer 1

Wenn bei der Division von Festkommazahlen einfacher Wortlänge mit Sicherheit feststeht, daß der Betrag des Dividenden gleich oder größer als der des Divisors ist, dann kann durch Anwendung des im allgemeinen bei Festkommazahlen doppelter Wortlänge (Abschnitt 4.6.) benutzten Befehls

DVD	n	Dividiere doppelt lang	$\langle A \rangle := \langle A, Q \rangle : \langle n \rangle$
-----	---	------------------------	---

vermieden werden, daß als Ergebnis eine Gleitkommazahl entsteht, bzw. ein BÜ-Alarm erfolgt.

Register A, Q

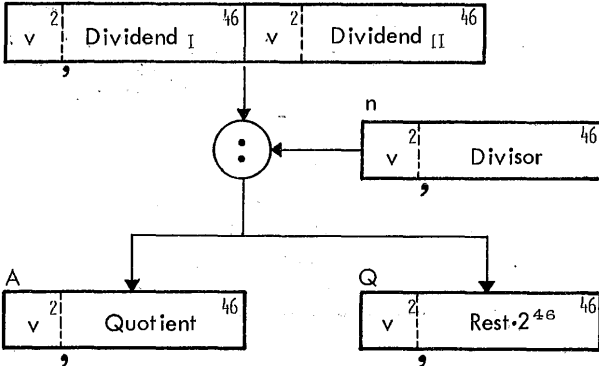


Bild 3.8 Beispiel zum Befehl DVD

Das doppelt lange Register AQ bietet dabei die Möglichkeit, durch einen Registertausch $\langle A \rangle := \langle Q \rangle$ den Dividenden so zu verändern, daß er auf jeden Fall kleiner als der Divisor wird. Nach der anschließenden Division mit dem Befehl DVD steht der ganzzahlige Teil des Ergebnisses in A, der Rest in Q. Durch einen weiteren Registertausch $\langle A \rangle := \langle Q \rangle$ wird der Rest nach A gebracht. Er kann nun mit Hilfe des Befehls DV dividiert werden.

Für die allgemeine Kommarechnung gilt, ausgehend vom Komma links und einer Kommaverschiebung

um x Stellen beim Dividend (vor dem Registertausch) und um y Stellen beim Divisor

eine Kommaverschiebung von

$x-y+(z)$ Stellen für den Quotienten
 $x-z+(z)$ Stellen für den Rest

(Länge des Registers: z Stellen)

*Durch den Registertausch $\langle A \rangle := \langle Q \rangle$ wurde das Komma für die Rechnung bereits um z Stellen nach links geschoben. Dieser Vorgang wird hier wieder rückgängig gemacht.

Positive Werte: Verschiebung nach rechts;
 Negative Werte: Verschiebung nach links.

$$800 : 021 = 000800 : 021 = 038 \text{ Rest } 002$$

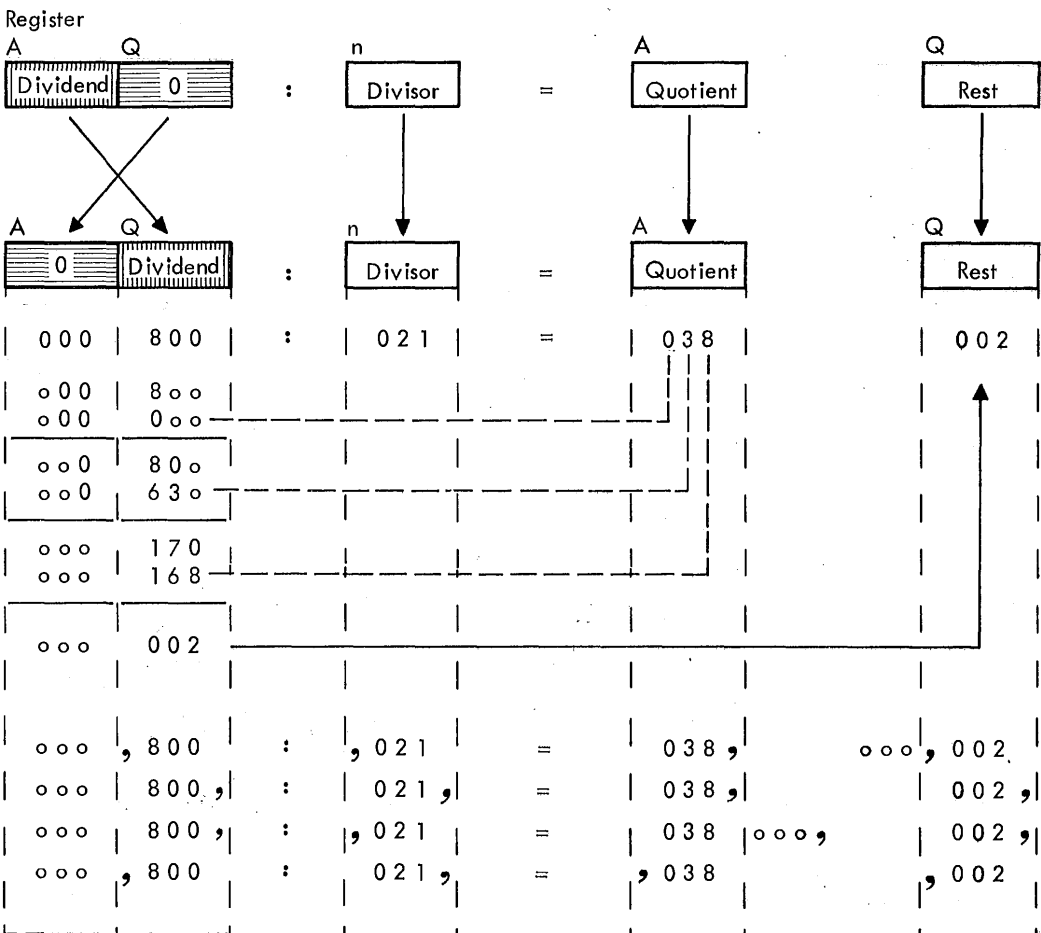
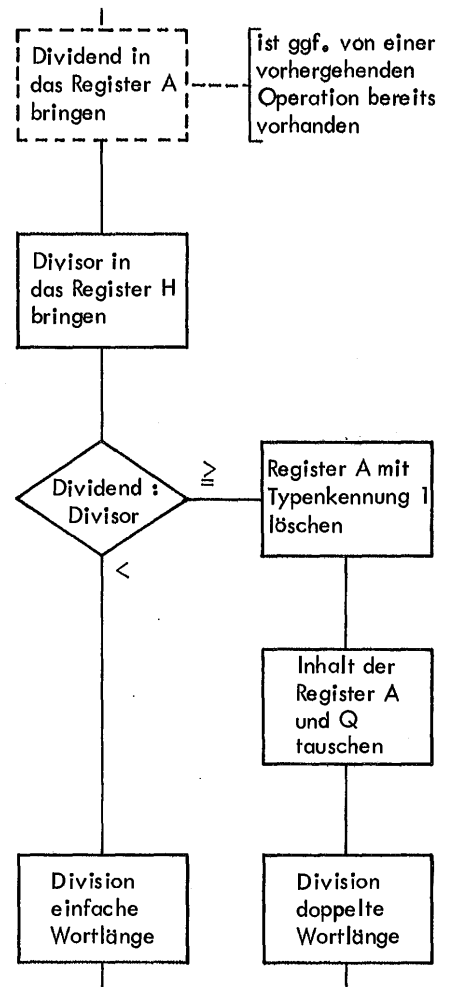
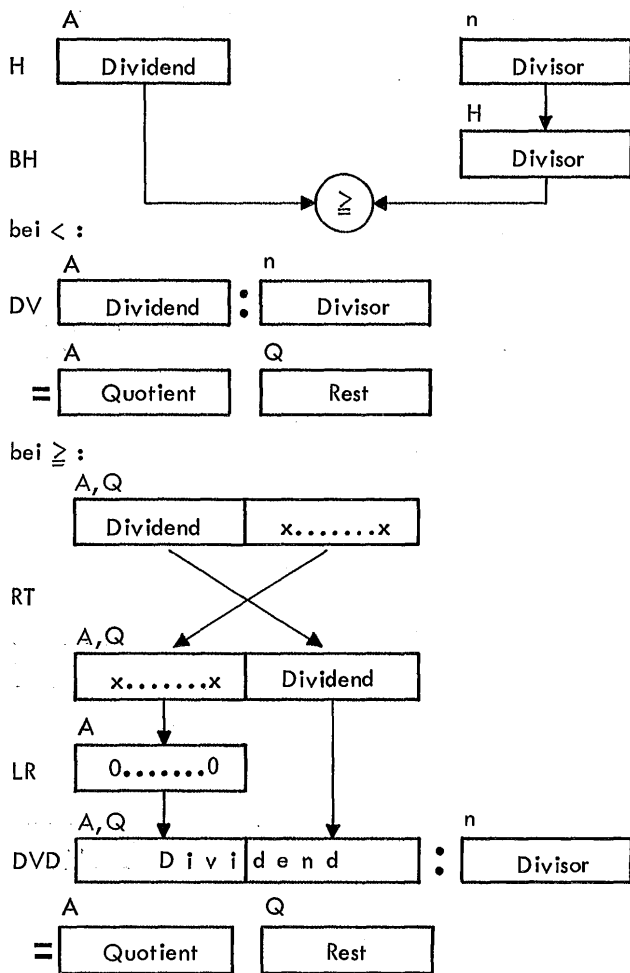


Bild 3.9 Dezimales Beispiel einer Division mit dem Befehl DVD bei Dividend gleich oder größer Divisor



3.5.3. Divisions-Ergebnis kleiner o d e r größer 1

Wenn die Größe der Operanden (Dividend und Divisor) unbekannt ist, dann wird programmäßig ebenso verfahren wie in Abschnitt 3.5.2. beschrieben; jedoch muß im Programm mit Hilfe eines Vergleichs zusätzlich festgestellt werden, ob der Dividend kleiner als der Divisor ist. Ergibt sich aus dieser Vergleichsabfrage, daß der Dividend kleiner als der Divisor ist, so wird die Division wie im Abschnitt 3.5.1. beschrieben durchgeführt, d.h. es wird auf die entsprechende Zeile im Programm (Division mit Befehl DV) gesprungen. Im anderen Fall erfolgt die Division mit dem Befehl DVD.



Name Code Adressenteil

```

.
.
.
B DIVIDEND,
BH DIVISOR,
SK DIVISION,
RT AQ,
LR A1,
DVD DIVISOR,
.
.
DIVISION= DV DIVISOR,
.
.

```

```

.
.
.
Bringe Dividend in das Register A
Bringe Divisor in das Register H
Springe wenn Dividend kleiner als Divisor nach "Division"
Tausche den Inhalt der Register A und Q
Lösche Register A mit Typenkennung 1
Dividiere doppelt langen Dividenden
.
.
Dividiere einfach langen Dividenden
.
.

```

Bild 3.10 Beispiel für eine Division wenn die Größe der Operanden unbekannt

3.5.4. Divisionsergebnis mit höherer Genauigkeit

Um ein möglichst genaues Divisionsergebnis zu erzielen, kann der bei einer Division evtl. entstehende Rest wiederum dividiert werden, und zwar so oft, bis die gewünschte Genauigkeit erreicht oder die Division aufgegangen ist. Die Quotienten werden dabei der Reihe nach hintereinander abgelegt.

Die Division selbst geht so vor sich, wie in den Abschnitten 3.5.1., 3.5.2. oder 3.5.3. bereits beschrieben. Da es sich bei den Divisionsresten stets um Zahlen < 1 handelt, wird deren Division mit Hilfe des Befehls DV durchgeführt.

Beispiel:

$$060 : 070 = 857 \text{ Rest } 010$$

$$010 : 070 = 142 \text{ Rest } 060$$

$$060 : 070 = 857 \text{ Rest } 010$$

$$060 : 070 = 857 \text{ 142 } 857 \text{ Rest } 010$$

Eine Kommarechnung ist nicht erforderlich, da der 1. Quotient die Kommastellung bestimmt.

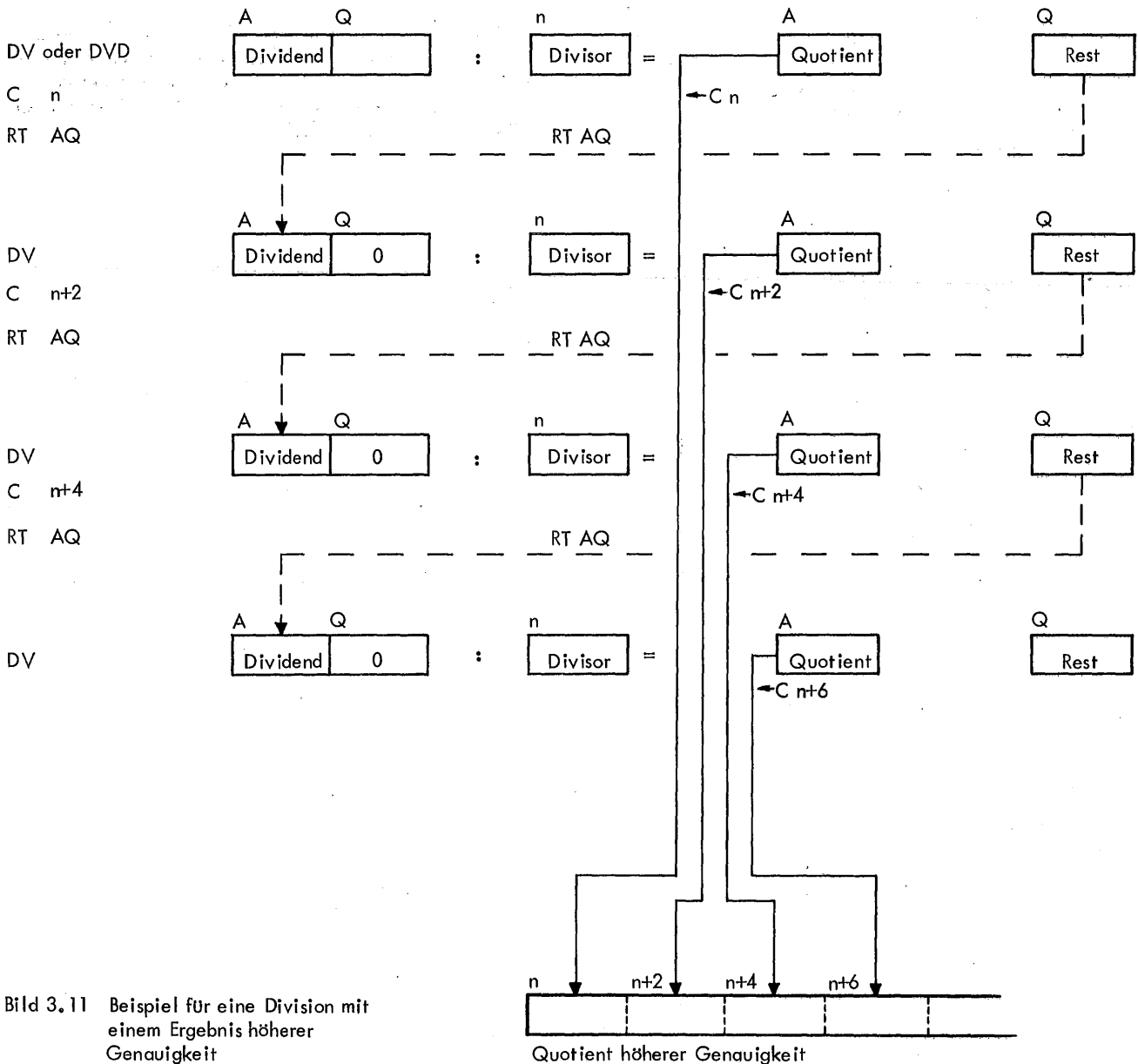


Bild 3.11 Beispiel für eine Division mit einem Ergebnis höherer Genauigkeit

H

IAS - Handbuch

Dez. 68

3.6. Schiften

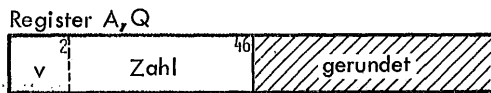
Das Schiften von Zahlwörtern wird mit dem Befehl

SH	s	p	Schifte	gemäß Spezifikation um p Stellen
----	---	---	---------	-------------------------------------

vorgenommen, wobei entweder im Register A oder im Register Q geschifft werden kann. Es ist auch möglich, in der Spezifikation beide Register anzugeben. Sie werden dann gleichzeitig, jedes für sich, auf die gleiche Art geschifft. Ist in der Spezifikation ein Z angegeben, so wird im doppelt langen Register A, Q geschifft.

Es kann rechts oder links geschifft werden. Ein Rechtsschift um x Stellen bedeutet bei Festkommazahlen eine Division durch 2^x , ein Linksschift um x Stellen eine Multiplikation mit 2^x . Läuft während des Linksschiftens die Zahl über, so wird der Schift zu Ende geführt und BÜ-Alarm gegeben. Das Vorzeichen wird in allen Fällen berücksichtigt (intern werden vorzeichengleiche Stellen nachgezogen).

Durch den Buchstaben R in der Spezifikation kann eine Rundung gewünscht werden. Beim Rechtsschift wird auf Grund der rechts herausgeschifteten Stellen eine Rundung der Zahl vorgenommen. Eine Rundung beim Linksschift ist nur dann von Bedeutung, wenn im doppelt langen Register A, Q geschifft wird. In diesem Fall wird nach dem Schift die Festkommazahl doppelter Wortlänge auf einfache Wortlänge gekürzt und gerundet (siehe Abschnitt 4.2.).



Ist die Spezifikation B angegeben, so werden die aus dem Register A herausgeschifteten L-Bits im Register Y gezählt.

Die Spezifikationen K (Kreisschift) und U (unabhängig von TK) sind für Festkommazahlen unbedeutend, da sie unabhängig von der Typenkennung ein Bitmuster schiften und auf das Vorzeichen der Festkommazahl keine Rücksicht nehmen.

Das Schiften ist im Abschnitt "Sonstige Operationen" speziell behandelt.

3.7. Umwandlung in eine Gleitkommazahl

Die Umwandlung einer Festkommazahl in eine Gleitkommazahl wird mit dem Befehl

NRM	FG	Normalisiere	<A> Gleitk. := <A, Q> Festk.
-----	----	--------------	------------------------------

bewirkt. Die Festkommazahl steht im doppelt langen Register A, Q, das Gleitkommaergebnis im Register A.

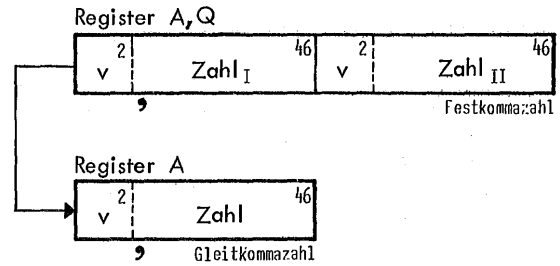


Bild 3.12 Beispiel für Umwandlung in Gleitkommazahl

Die Typenkennung der Festkommazahl muß 1 sein, das Ergebnis erhält die Typenkennung 0.

Die Vorzeichen der beiden Register müssen gleich sein. Zu diesem Zweck sollte vorher der Befehl VAQ gegeben werden.

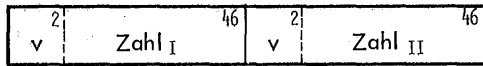
Das Register Q darf nicht über- oder untergelaufen sein. Ein über- oder untergelaufenes Zahlwort im Register A wird richtig verarbeitet.

Steht das Komma links, so ist das Ergebnis richtig. Ist das Komma um x Stellen nach rechts verschoben, so muß das Ergebnis mit dem Faktor 2^x multipliziert werden.

Soll eine Festkommazahl einfacher Wortlänge umgewandelt werden, so ist zunächst das Register Q mit dem Befehl LR 1Q zu löschen. Wenn das Vorzeichen der Festkommazahl negativ sein kann, sind mit dem Befehl VAQ die Vorzeichen anzugleichen.

4. OPERATIONEN FÜR DOPPELTE WORTLÄNGE

Festkommazahlen mit doppelter Wortlänge sind in zwei Registern bzw. in zwei aufeinanderfolgenden Speicherzellen untergebracht. Sie werden intern in einigen Fällen getrennt verarbeitet.



Die Zahlen bestehen also aus zwei Teilen (I und II) und haben eigene Vorzeichen. Sie können vom Programmierer als eine Zahl betrachtet werden. Dabei ist jedoch auf eine Besonderheit zu achten:

4.1. Vorzeichenangleich

Bei der Addition und Subtraktion können die beiden Vorzeichen (je 2 Bits) ungleich sein. Sie werden jedoch von arithmetischen Festkommabefehlen (Addition, Subtraktion, Multiplikation und Division) richtig verarbeitet. Bevor jedoch eine Verarbeitung der doppelt langen Festkommazahl durch einen anderen Befehl erfolgt, müssen die Vorzeichen durch den Befehl VAQ angeglichen werden. Dieser Befehl berichtigt außerdem das Ergebnis unter Erhaltung des Zahlenwertes. Die Kommastellung bleibt unverändert.

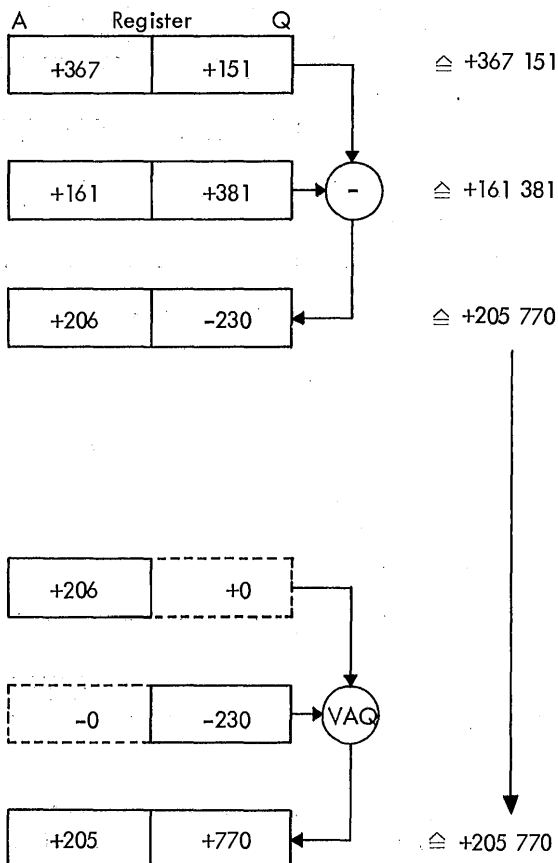


Bild 4.1 Beispiel für Vorzeichenangleich mit VAQ

4.2. Verlängerung und Verkürzung

Eine Festkommazahl einfacher Wortlänge kann auf doppelte Wortlänge verlängert werden, indem man sie rechts oder links um eine Wortlänge erweitert.

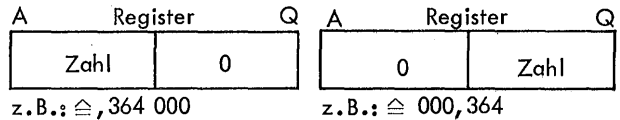


Bild 4.2 Beispiel für Verlängerung und Verkürzung

Zu diesem Zweck muß die Zahl (bei Verlängerung rechts) im Register A oder (bei Verlängerung links) im Register Q stehen. Jeweils das andere Register (Q bzw. A) wird mit Typenkennung 1 gelöscht. Gegebenenfalls müssen durch den Befehl VAQ die Vorzeichen angeglichen werden.

Wird eine Festkommazahl doppelter Wortlänge in eine Zahl einfacher Wortlänge umgewandelt, so kann entweder die rechte oder die linke Hälfte entfallen. Vorher müssen durch den Befehl VAQ die Vorzeichen angeglichen werden. Soll die rechte Hälfte fortfallen, so werden auf die Zahl nur diejenigen Befehle angewandt, die ausschließlich das Register A als Operand haben. Soll dagegen die Verkürzung durch Rundung erfolgen, so sind ebenfalls zuerst mit dem Befehl VAQ die Vorzeichen anzugleichen. Anschließend wird dann mit dem Befehl SH ZLR 0 die Rundung durchgeführt.

Will man die Zahl um die linke Hälfte verkürzen, so sind ebenfalls zunächst die Vorzeichen anzugleichen. Mit Hilfe des Befehls SH ZL 46 wird dann die Zahl um 46 Stellen nach links geschiftet. Enthält die linke Hälfte Stellen, die nicht Null sind, so erfolgt BÜ-Alarm. Die Festkommazahl einfacher Wortlänge steht im Register A.

Die Kommastelle der Zahl bleibt unverändert.

4.3. Transporte

Für das Bringen einer Festkommazahl doppelter Wortlänge stehen die Befehle zur Verfügung:

BZ	n	Bringe zwei Wörter	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n+2 \rangle$
BZN	n	Bringe zwei Wörter negativ	$\langle A \rangle := -\langle n \rangle$ $\langle Q \rangle := -\langle n+2 \rangle$

Ist eines der Speicherwörter markiert, so wird das Markenregister auf L gesetzt und im Register das 1. Bit dem 2. Bit angeglichen. Damit sind die beiden Vorzeichenstellen gleich. Dieser Vorgang wird durch die Gleichung

$$\langle M \rangle := \text{Marke}$$

ausgedrückt.

H

TAS - Handbuch

Dez. 68

Für das Speichern steht der Befehl

CZ	n	Speichere zwei Wörter	$\langle n \rangle := \langle A \rangle$ $\langle n+2 \rangle := \langle Q \rangle$
----	---	-----------------------	--

zur Verfügung. Die Festkommazahl darf nicht über- oder untergelaufen sein, d.h. im Register A (wie auch im Register Q) müssen die beiden Vorzeichenbits gleich sein.

Sind die beiden Vorzeichenbits im Register nicht gleich, dann wird dieses Register nicht abgespeichert. Es erfolgt BÜ-Alarm.

4.4. Addition - Subtraktion

Zu einer Festkommazahl doppelter Wortlänge kann eine Festkommazahl einfacher Wortlänge hinzuaddiert (bzw. subtrahiert) werden.

AQ	n	Addiere in Q	$\langle A, Q \rangle := \langle A, Q \rangle + \langle n \rangle$
SBQ	n	Subtrahiere in Q	$\langle A, Q \rangle := \langle A, Q \rangle - \langle n \rangle$

Der Inhalt beider Register wie auch derjenige der Speicherzelle muß die Typenkennung 1 haben. Die beiden Register dürfen nicht über- oder untergelaufen sein. Das Vorzeichen kann jedoch in den beiden Registern verschieden sein.

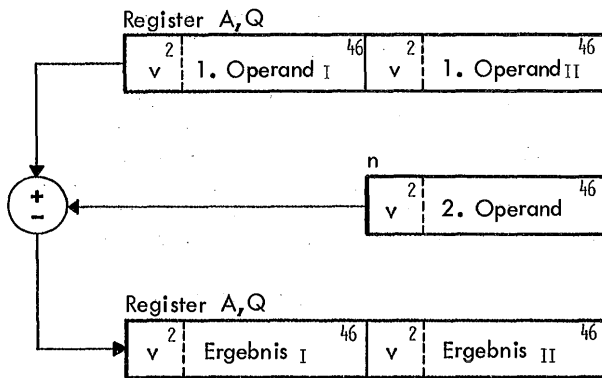


Bild 4.3 Beispiel zu den Befehlen AQ und SBQ

Der 1. Operand muß im doppelt langen Register A,Q stehen, der zweite befindet sich in der Speicherzelle n. Das Ergebnis steht wiederum im doppelt langen Register A,Q. Die Vorzeichen des Ergebnisses können verschieden sein.

Für die einfache Kommaechnung gilt:

1. Operand	2. Operand	Ergebnis
rechts	rechts	rechts
Mitte	links	Mitte

Das Komma kann zwar (von rechts ausgehend) beim 1. und 2. Operanden an beliebiger Stelle stehen, jedoch muß die Kommastellung bei beiden Operanden gleich sein.

Mit den Multiplikationsbefehlen

MLA	n	Multipliziere akkumulierend	$\langle A, Q \rangle := \langle A \rangle \cdot \langle n \rangle + \langle H, Q \rangle$
MAN	n	Multipliziere akkumulierend negativ	$\langle A, Q \rangle := -\langle A \rangle \cdot \langle n \rangle + \langle H, Q \rangle$

kann gleichzeitig zu dem doppelt langen Produkt ein doppelt langer Summand addiert werden. Weitere Einzelheiten zur Multiplikation sind im Abschnitt 3.4.1. aufgeführt.

Die Typenkennung der Operanden muß in allen Fällen 1 sein. Das Ergebnis hat wiederum die Typenkennung 1.

Die Vorzeichen bei den doppelt langen Festkommazahlen (Produkt, Summand und Ergebnis) können für jedes Register verschieden sein; sie werden richtig verarbeitet.

Alle Festkommazahlen sollten jedoch nicht über- oder untergelaufen sein, da sonst ein genaues Ergebnis nicht gewährleistet ist. Ist das Ergebnis über- oder untergelaufen, so wird BÜ-Alarm gegeben.

4.5. Multiplikation

Bei der Multiplikation von Festkommazahlen einfacher Länge entsteht bei den Befehlen

ML	n	Multipliziere	$\langle A, Q \rangle := \langle A \rangle \cdot \langle n \rangle$
MLN	n	Multipliziere negativ	$\langle A, Q \rangle := -\langle A \rangle \cdot \langle n \rangle$
MLA	n	Multipliziere akkumulierend	$\langle A, Q \rangle := \langle A \rangle \cdot \langle n \rangle + \langle H, Q \rangle$
MAN	n	Multipliziere akkumulierend negativ	$\langle A, Q \rangle := -\langle A \rangle \cdot \langle n \rangle + \langle H, Q \rangle$

ein Ergebnis von doppelter Wortlänge. Diese Befehle sind bereits im Abschnitt 3.4.1. erläutert.

4.6. Division

Für die Division einer doppelt langen Festkommazahl ist der Befehl

DVD	n	Dividiere doppelt lang	$\langle A \rangle := \langle A, Q \rangle : \langle n \rangle$
-----	---	------------------------	---

vorgesehen.

Der Divisor darf nicht Null sein, da sonst der Quotient eine Gleitkommazahl wird, die normalisiert, aber nicht gerundet ist. Das Ergebnis ist nur dann stellenrichtig, wenn Dividend und Divisor das Komma links haben. Es wird BÜ-Alarm gegeben.

Beim Dividieren können die Vorzeichen verschieden sein. Sie werden - wie beim Befehl VAQ - angeglichen. Nach dem Angleich muß der Betrag des Dividenden kleiner sein als der des Divisors (bei Komma links).

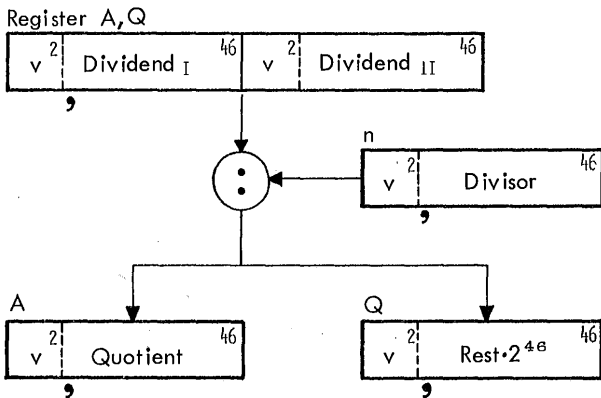


Bild 4.4 Beispiel zum Befehl DVD

Der Dividend muß im doppelt langen Register A,Q stehen. Der Divisor wird aus der Speicherzelle n geholt. Der Quotient steht im Register A und der Divisionsrest im Register Q. Wird bei allen Zahlen das Komma links angenommen, dann ist der Rest um den Faktor 2^{46} zu groß.

Dividend und Divisor müssen die Typenkennung 1 haben. Beide dürfen nicht unter- oder übergelaufen sein.

Für die einfache Kommarechnung ergibt sich folgende Übersicht über die möglichen Änderungen:

Dividend	Divisor	Quotient	Rest $\cdot 2^{46}$
links	links	links	links
mitte	links	rechts	rechts
mitte	rechts	links	rechts

Für die allgemeine Kommarechnung gilt, ausgehend vom Komma links und einer Kommaverschiebung

um x Stellen beim Dividend und
um y Stellen beim Divisor

eine Kommaverschiebung von

x-y Stellen für Quotienten
x-46 Stellen für den Rest.

Positive Zahlen bedeuten Verschiebung nach rechts, negative eine Verschiebung nach links. Für den Gleitkommafall muß der Gleitkommaquotient mit dem Faktor 2^{x-y} multipliziert werden. Die Umwandlung einer Gleitkommazahl in eine Festkommazahl ist im Abschnitt 3.7. beschrieben.

4.7. Schiften

Festkommazahlen doppelter Wortlänge müssen, bevor sie geschifft werden, gleiche Vorzeichen haben. Falls die Vorzeichen ungleich sein können, sollten sie mit dem Befehl VAQ angeglichen werden. Geschifft wird mit dem Befehl

SH	Zsp	Schifte	Schift (zusammen) gemäß Spezifikation um p Stellen

Ein Schift nach rechts um p Stellen bedeutet eine Division der Festkommazahl durch 2^p . Beim Linksschift ist für "s" der Buchstabe L anzugeben. Die Festkommazahl wird dann mit dem Faktor 2^p multipliziert. Das Ergebnis ist vorzeichenrichtig.

Intern gesehen werden die Vorzeichenstellen vom Register Q umschifft und links bzw. rechts vorzeichen-gleiche Stellen nachgezogen.

Durch Angabe des Buchstabens R wird nach dem Schiften eine Rundung durchgeführt. Beim Rechtsschift wird nach dem Schift in Abhängigkeit von den herausgeschifteten Stellen die Festkommazahl gerundet.

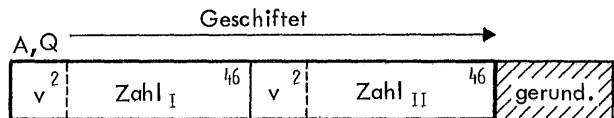


Bild 4.5 Beispiel für Rechtsschift mit Rundung

Bei einem Linksschift wird dagegen nach dem Schiften die doppelt lange Festkommazahl auf einfache Wortlänge gekürzt und gerundet.

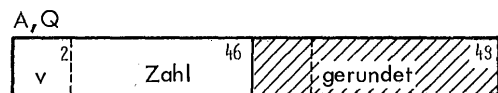


Bild 4.6 Beispiel für Linksschift mit Rundung

Läuft während des Schiftens nach links die Zahl über, so wird der Schift zu Ende geführt, und es gibt BÜ-Alarm (siehe auch Abschnitt 4.2.).

Wird die Spezifikation B verwendet, so werden die aus dem Register A herausgeschifteten L-Bits im Register Y gezählt.

Die Spezifikationen K und U sind für Festkommazahlen unbedeutend, da sie unabhängig von der Typenkennung um ein Bitmuster schiften und auf das Vorzeichen der Festkommazahl keine Rücksicht nehmen.

Sind die Vorzeichen nicht gleich, so wird beim Linksschift das Register A und beim Rechtsschift das Register Q invertiert. Dabei verändert sich der Wert der Festkommazahl.

Das Schiften ist im Abschnitt "Sonstige Operationen" speziell behandelt.



TAS - Handbuch

Juli 69

5. OPERATIONEN FÜR HALBE WORTLÄNGE

Eine Festkommazahl halber Wortlänge hat 24 Binärstellen. Beim Rechnen mit Zahlen halber Wortlänge sind - wie bei einfacher Wortlänge - zwei Bits für das Vorzeichen vorgesehen. Über- oder untergelaufene Zahlen werden zum Teil verarbeitet, können aber zu falschen Ergebnissen führen. Eine Marke besitzen Halbwörter nicht.

5.1. Transport

Für das Bringen von Operanden ins Rechenwerk (zur Ausführung arithmetischer Operationen) sind die Befehle

B2V	m	Bringe Halbwort mit Vorzeichen	$\langle A \rangle := \langle m \rangle$
B2VN	m	Bringe Halbwort mit Vorzeichen negativ	$\langle A \rangle := -\langle m \rangle$

vorgesehen. Der Operand wird vorzeichenrichtig ins Register A gebracht. Die Befehle B2 und B2Z bringen lediglich ein Bitmuster und berücksichtigen die Vorzeichen nicht.

Zum Abspeichern der Ergebnisse in den Speicher steht der Befehl

G2	m	Speichere Halbwort	$\langle m \rangle := \langle A \rangle_{25-48}$
----	---	--------------------	--

zur Verfügung. Die rechte Hälfte des Registers A wird unverändert in die Speicherzelle m gebracht. Die Zahl wird nicht auf Über- oder Unterlauf geprüft.

5.2. Addition - Subtraktion

Für die Addition bzw. Subtraktion stehen die Befehle

A2	m	Addiere Halbwort	$\langle A \rangle := \langle A \rangle_{25-48} + \langle m \rangle$
SB2	m	Subtrahiere Halbwort	$\langle A \rangle := \langle A \rangle_{25-48} - \langle m \rangle$

zur Verfügung. Beide Operanden sollen nicht über- oder untergelaufen sein, da sonst das Ergebnis falsch sein kann, ohne daß ein BÜ-Alarm gegeben wird.

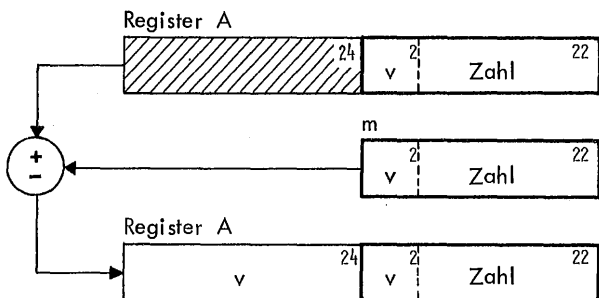


Bild 5.1 Beispiel zu den Befehlen A2 und SB2

Ist das Ergebnis über- oder untergelaufen, so wird BÜ-Alarm gegeben. Das Ergebnis kann, wenn die Operanden nicht über- oder untergelaufen waren, auch als Festkommazahl einfacher Wortlänge verwendet werden (linke Hälfte des Registers ist vorzeichengleich gesetzt worden).

Die Typenkennung kann beliebig sein, während die Kommastellung zwar ebenfalls beliebig ist, jedoch bei beiden Operanden übereinstimmen muß.

5.3. Multiplikation

Für die Multiplikation von Festkommazahlen halber Wortlänge stehen zwei Befehlsgruppen zur Verfügung. In beiden Fällen ist das Ergebnis eine Festkommazahl einfacher Wortlänge. Die nachstehenden Befehle eignen sich besonders, um das Vielfache einer Zahl zu bilden.

M2	m	Multipliziere Halbwort	$\langle A \rangle := \langle A \rangle \cdot \langle m \rangle$
M2N	m	Multipliziere Halbwort negativ	$\langle A \rangle := -\langle A \rangle \cdot \langle m \rangle$

Der Operand im Register kann eine Festkommazahl einfacher oder halber Wortlänge sein, der Operand im Speicher hat halbe Wortlänge.

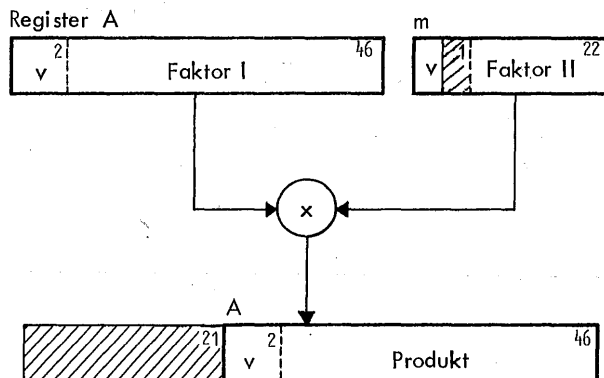


Bild 5.2 Beispiel zu den Befehlen M2 und M2N

Das Ergebnis hat einfache Wortlänge. Ist es über- oder untergelaufen, so wird BÜ-Alarm gegeben.

Ist der Faktor I übergelaufen, so wird er trotzdem weiter verarbeitet; es kann jedoch ein falsches Ergebnis entstehen. Faktor II darf nicht übergelaufen sein. Näheres ist dem Befehls-Lexikon zu entnehmen.

Für die einfache Kommarechnung gilt:

Faktor ($\langle A \rangle$)	Faktor ($\langle n \rangle$)	Produkt
rechts	rechts	rechts
links	rechts	links

Für die allgemeine Kommarechnung gilt - ausgehend vom Komma rechts -

Komma beider Faktoren um x bzw. y Stellen nach links verschoben,
Komma des Produkts um $x + y$ Stellen nach links verschoben.

Die Typenkennung ist beliebig; es wird jedoch stets wie bei Festkommazahlen multipliziert.

Falls durch einen über- oder untergelaufenen Operanden ein über- oder untergelaufenes Ergebnis entsteht, wird ein BÜ-Alarm gegeben. Das Ergebnis kann jedoch falsch sein.

Bei den nachstehenden Befehlen wird das Ergebnis gerundet. Sie eignen sich besonders, um Teile einer Zahl zu bilden (z.B. Prozente einer Zahl).

M2R	m	Multipliziere Halbwort mit Rundung	$\langle A \rangle := \langle A \rangle \cdot \langle m \rangle$
M2NR	m	Multipliziere Halbwort negativ mit Rundung	$\langle A \rangle := -\langle A \rangle \cdot \langle m \rangle$

Der Operand im Register kann eine Festkommazahl einfacher oder halber Wortlänge sein, der Operand im Speicher hat halbe Wortlänge.

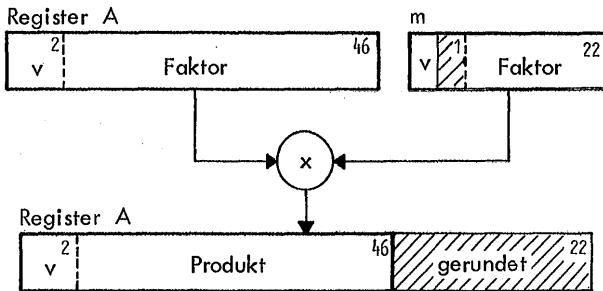


Bild 5.3 Beispiel zu den Befehlen M2R und M2NR

Das Ergebnis hat einfache Wortlänge und ist gerundet.

Für die einfache Kommarechnung gilt

Faktor ($\langle A \rangle$)	Faktor ($\langle n \rangle$)	Produkt
links	links	links
rechts	links	rechts

Für die allgemeine Kommarechnung gilt - ausgehend vom Komma rechts -

- bei einer Kommaverschiebung um x bzw. y Stellen nach rechts bei den Faktoren:
- eine Kommaverschiebung um x + y Stellen beim Produkt.

Die Typenkennung der Operanden muß in allen Fällen 1 sein. Das Ergebnis hat wiederum die Typenkennung 1.

6. TEILWÖRTER

Bei den Teilwortbefehlen BT, CT, AT und SBT werden aus einem Ganzwort mit Hilfe einer Marke einzelne Binärstellen herausgenommen, die dann verarbeitet werden. Weitere Einzelheiten sind im Abschnitt "Nichtzahlwörter" beschrieben.

H

TELEFUNKEN
COMPUTER

K

TAS - HANDBUCH

Gleitkomma-Arithmetik

INHALT

GLEITKOMMA - ARITHMETIK

<u>1. EXTERNE DARSTELLUNG</u>	steht noch aus
<u>2. INTERNER AUFBAU</u>	2 - 1
2.1. Bereich	2 - 1
2.2. Genauigkeit	2 - 2
2.3. Vorzeichen	2 - 2
2.4. Gleitkomma-Null	2 - 3
2.5. Normalisieren - Denormalisieren	2 - 3
2.6. Bereichüberschreitung	2 - 4
2.7. Rundung	2 - 4
2.8. Vergleiche	2 - 5
2.9. Betrag	2 - 5
2.10. Typenkennung	2 - 5
2.11. Marke	2 - 6
2.12. Einerrücklauf	2 - 7
<u>3. OPERATIONEN FÜR EINFACHE WORTLÄNGE</u> ...	3 - 1
3.1. Bringen	3 - 1
3.2. Speichern	3 - 2
3.3. Addition - Subtraktion	3 - 4
3.4. Multiplikation	3 - 6
3.5. Division	3 - 7
<u>4. OPERATIONEN FÜR DOPPELTE WORTLÄNGE</u>	4 - 1
4.1. Bringen	4 - 1
4.2. Speichern	4 - 1
4.3. Addition - Subtraktion	4 - 1
4.4. Multiplikation	4 - 1
<u>5. KONVERTIEREN</u>	steht noch aus



2. INTERNER AUFBAU

Gleitkommazahlen werden intern durch eine Mantisse und einen Exponenten zur Basis 16 dargestellt. Sie hat also die Form

$$m \cdot 16^e$$

wobei m die Mantisse und e der Exponent ist. Die Mantisse wird stets als echter Bruch dargestellt, das Komma steht also ganz links. Der Exponent besteht aus 7 Bits und einem Bit für das Vorzeichen. Die Länge der Mantisse kann 38 Bits oder 84 Bits umfassen, zuzüglich der Vorzeichenstellen. Aus der Länge der Mantisse ergeben sich zwei Darstellungen der Gleitkommazahlen

einfache Wortlänge: 48 Bits
 doppelte Wortlänge: 96 Bits

Im Bild 2.1. ist der interne Aufbau gezeigt, wie er bei der Verarbeitung im Rechenwerk auftritt. Im Speicher nimmt eine Vorzeichenstelle (Bit 1) das Markenbit auf (siehe dazu Abschnitt 2.11.).

verwendet. Der Betrag einer normalisierten Gleitkommazahl kann maximal den Wert

$$\begin{aligned} & (1 - 2^{-38}) \cdot 16^{+127} \\ & = 2^{508} - 2^{470} \\ & > 0,837\ 987\ 995\ 6 \dots \cdot 10^{+153} \\ & > 0,8 \cdot 10^{+153} \end{aligned}$$

annehmen. Beim kleinsten Betrag einer normalisierten Gleitkommazahl muß eines der Bits 3 bis 7 vom Vorzeichen verschieden sein; sodaß der minimalste Wert

$$\begin{aligned} & 2^{-4} \cdot 16^{-127} \\ & = 2^{-512} \\ & < 0,745\ 834\ 073\ 2 \dots \cdot 10^{-154} \\ & < 0,7 \cdot 10^{-154} \end{aligned}$$

beträgt. Der Betrag einer Gleitkommazahl kann sich also innerhalb dieser Bereiche bewegen.

$$0,745 \dots \cdot 10^{-154} < |x| < 0,837 \dots \cdot 10^{+153}$$

2. 1. Bereich

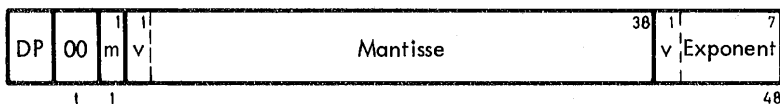
Der Zahlenbereich, den die Gleitkommazahlen übertreten können, ist im wesentlichen durch den Exponenten bestimmt. Für die nachstehenden Betrachtungen sollen normalisierte Gleitkommazahlen angenommen werden und da die negativen Zahlen ein Spiegelbild der positiven Zahlen sind, wurden die Beträge der Gleitkommazahlen

Damit können auf der Zahlengerade beiderseits des Nullpunktes 155 Dezimalstellen erreicht werden. Dies macht die Gleitkomma-Arithmetik besonders für solche Aufgabengebiete geeignet, bei denen mit Zahlen von äußerst unterschiedlicher Größenordnung gerechnet werden muß.

Bei Gleitkommazahlen doppelter Wortlänge gelten die

GLEITKOMMAZAHN (Basis 16)

Einfache Länge



Doppelte Länge

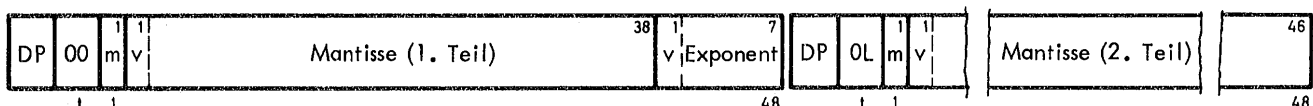


Bild 2.1 Interne Darstellung der Gleitkommazahlen

gleichen Werte. Lediglich der maximal darstellbare Wert erhöht sich gering

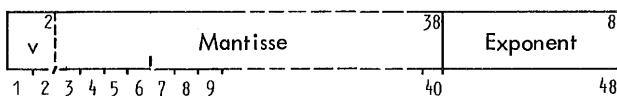
$$\begin{aligned}
 & (1-2^{-34}) \cdot 16^{+127} \\
 & = 2^{508} - 2^{424} \\
 & > 0,837\ 987\ 995\ 621\ 412\ 318\ 723\ 376 \dots \cdot 10^{153} \\
 & > 0,8 \cdot 10^{+153}
 \end{aligned}$$

Der minimal darstellbare Wert ist identisch mit dem der einfachen Wortlänge.

2. 2. Genauigkeit

Bei normalisierten Gleitkommazahlen ist es von der Zahl abhängig, ob die linken 3 Stellen der Mantisse bedeutsame Stellen sind oder ob sie zur Genauigkeit nicht mehr beitragen. Dadurch ist die Genauigkeit von der darzustellenden Zahl abhängig. Im Bild 2.2. ist im Prinzip gezeigt, wie die Genauigkeit schwankt. Für die Gleitkommazahl einfacher Länge schwankt daher die Genauigkeit zwischen 35 und 38 Binärstellen. Obwohl über die Hälfte der möglichen Zahlen eine Genauigkeit von 38 Binärstellen hat, soll hier die garantierte Genauigkeit von 35 Binärstellen zugrunde gelegt werden.

Zahl dezimal	Darstellung dezimal	Mantisse intern	Genauigkeit in Binärstellen
2^{4n}	$2^{-4} \cdot 16^{n+1}$,000LO...	35
2^{4n+1}	$2^{-3} \cdot 16^{n+1}$,00L00...	36
2^{4n+2}	$2^{-2} \cdot 16^{n+1}$,0L000...	37
2^{4n+3}	$2^{-1} \cdot 16^{n+1}$,L0000...	38
2^{4n+4}	$2^{-4} \cdot 16^{n+2}$,000LO...	35



eine der 4 Binärstellen muß verschieden dem Vorzeichen sein

Bild 2.2 Zur Genauigkeit einer normalisierten Gleitkommazahl

Die Genauigkeit einer Gleitkommazahl einfacher Wortlänge beträgt

$$\begin{aligned}
 & 2^{35} \\
 & > 3 \cdot 10^{10} \\
 & > 10 \text{ Dezimalstellen}
 \end{aligned}$$

Diese Genauigkeit wird auch bei der arithmetischen Operation erreicht, soweit dies nach den mathematischen Gesetzen möglich ist. Dies wird dadurch erreicht, daß intern der Exponententeil abgetrennt wird und der Mantissenteil auf 48 Bits erweitert wird.

Mit dem Mantissenteil wird dann wie mit Gleitkommazahlen gerechnet. Bei Multiplikationen ist das Zwischenergebnis doppelt lang. Am Ende der Operation wird der Exponent wieder eingesetzt.

Bei Gleitkommazahlen doppelter Wortlänge stehen für die Mantisse 84 Bits und Vorzeichenbits zur Verfügung, während für den Exponenten ebenfalls 7 Bits und 1 Vorzeichenbit zur Verfügung stehen. Das ergibt eine Erhöhung der Genauigkeit, während der Bereich nur unwesentlich vergrößert wird. Die garantierte Genauigkeit beträgt

$$\begin{aligned}
 & 2^{81} \\
 & > 2 \cdot 10^{24} \\
 & > 24 \text{ Dezimalstellen}
 \end{aligned}$$

Damit ist die Genauigkeit der doppelt langen Gleitkommazahl mehr als doppelt so hoch wie bei der einfachen Länge.

2. 3. Vorzeichen

Bei der Gleitkommazahl hat die Mantisse zwei Vorzeichenstellen und der Exponent eine Vorzeichenstelle. Sind die Vorzeichenstellen auf "0" gesetzt, so sind sie positiv, sind sie auf "L" gesetzt, so sind sie negativ.

Eine positive Zahl wird in eine negative Zahl verwandelt, indem jedes Bit seinen umgekehrten Wert erhält - die Zahl wird also invertiert (B-1-Komplement).

Beispiel:

$$\begin{aligned}
 0 \text{ L00L} & = +9 \\
 1 \text{ OLL0} & = -9
 \end{aligned}$$

Eine Subtraktion wird immer so durchgeführt, daß der Subtrahend invertiert und anschließend addiert wird. In

diesem Zusammenhang kann sich eine positive (${}_0 0000$) oder eine negative (${}_L LLLL$) Null ergeben. Eine negative Null hat nur interne Bedeutung und tritt beim Rechnen nach außen hin nicht in Erscheinung. Bei speziellen Operationen ist es jedoch von Bedeutung, daß beim Ergebnis Null oft die negative Null entsteht.

Die Mantisse hat zwei Vorzeichenstellen. Im Rechenwerk sind beide Vorzeichenstellen gleich. Die rechte Vorzeichenstelle wird für den Über- bzw. Unterlauf der Mantisse verwendet, hat aber nur interne Bedeutung. Im Speicher nimmt die linke Vorzeichenstelle die Marke auf (siehe Abschnitt 2.11.).

Der Exponent hat nur eine Vorzeichenstelle.

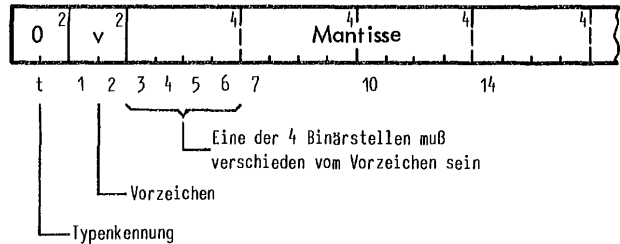


Bild 2.3 Normalisierte Gleitkommazahl

2.4. Gleitkomma-Null

Bei der Zahl Null kann die Mantisse sowohl plus Null (${}_{00} 0...0$) als auch minus Null (${}_{LL} L...L$) sein. Für die arithmetischen Operationen sind beide gleichwertig. Der Exponent ist dabei bedeutungslos.

Ist das Ergebnis einer Operation Null, so wird immer die normalisierte Gleitkomma-Null erzeugt. Es ist dies

$$- 0 \cdot 16^{-127}$$

Daraus ist zu schließen, daß in fast allen Fällen bei arithmetischen Operationen mit der normalisierten Gleitkomma-Null gerechnet wird.

2.5. Normalisieren - Denormalisieren

Eine Gleitkommazahl heißt normalisiert, wenn sie

- ungleich Null ist und wenn die ersten 4 Mantissenbits nicht alle gleich dem Vorzeichenbit sind. Hat dabei der Exponententeil den Wert 0, so kann er die positive oder die negative Null sein.
- die negative Null als Mantisse und -127 als Exponententeil hat.

Damit ist gewährleistet, daß das Wort gut ausgenutzt ist und wenig unbedeutende Stellen hat.

Gleitkommazahlen werden in den meisten Fällen in der normalisierten Form dargestellt. Mit Ausnahme der Befehle AU und SBU (Addiere bzw. Subtrahiere unnormalisiert) werden normalisierte Operanden vorausgesetzt. Das Ergebnis ist dann ebenfalls normalisiert und gerundet. Sind die Operanden nicht normalisiert, so kann nicht damit gerechnet werden, daß das Ergebnis normalisiert und gerundet ist.

Wird eine Gleitkommazahl normalisiert, so wird die Mantisse um jeweils 4 Binärstellen nach links geschiftet (mit 2^4 multipliziert), bis die Zahl normalisiert ist und dabei wird jeweils der Exponent um 1 verringert (die Zahl mit 16^{-1} multipliziert bzw. durch 16^{-1} dividiert). Damit bleibt bei einer Normalisierung der Wert der Zahl erhalten; es fallen jedoch die linken unbedeutenden Stellen fort. Ggfs. werden vorzeichengleiche Stellen nachgezogen.

Da bei den Operationen die Zwischenergebnisse eine verlängerte Mantisse haben, werden beim Normalisieren rechts bedeutsame Stellen nachgezogen und gehen damit nicht verloren. Bild 2.4. veranschaulicht dies.

Während bei der Addition, Subtraktion und Division voll normalisiert wird, wird bei der Multiplikation maximal um 1 Schritt = 4 Binärstellen normalisiert. Wenn die Operanden normalisiert sind, ist dies in jedem Falle ausreichend. Bei nicht normalisierten Operanden kann das Ergebnis nicht normalisiert sein.

Den umgekehrten Vorgang des Normalisierens wollen wir mit Denormalisieren bezeichnen. Er wird z.B. benötigt werden, um bei der Addition die Exponenten gleich zu machen und um den Über- bzw. Unterlauf der Mantisse zu beseitigen. Hierbei wird um jeweils 4 Stellen nach rechts geschiftet und dabei der Exponent erhöht. Das Denormalisieren ist weitgehend eine interne Angelegenheit.

Beim Normalisieren wird die Anzahl der Binärstellen, um die normalisiert wurde, in das Register Y gebracht. Mit Hilfe des Befehls R ist dieser Wert zugänglich. Entsteht als Ergebnis der Wert Null, so wird die Gleitkomma-Null erzeugt und das Register Y auf den Wert 0 gesetzt.

Mit dem Befehl NRM kann eine Gleitkommazahl normalisiert werden.

Für Gleitkommazahlen doppelter Länge gilt sinngemäß das gleiche.

K

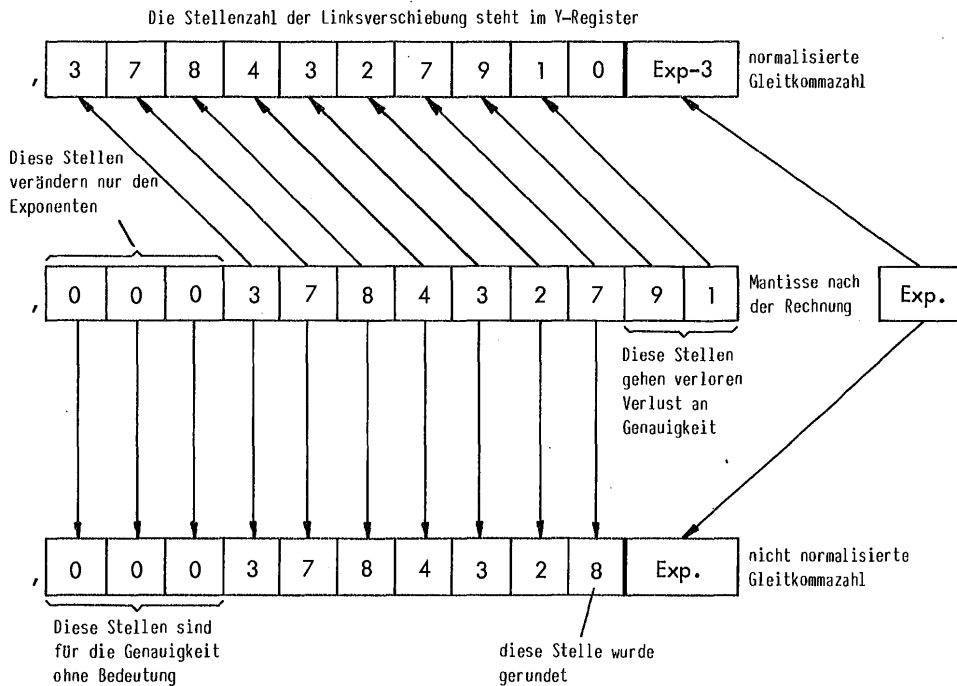


Bild 2.4 Genauigkeitsgewinn durch Normalisierung

2. 6. Bereichsüberschreitung

Wird der Betrag der Mantisse beim Zwischenergebnis gleich oder größer 1, so wird das rechte Vorzeichenbit einen anderen Wert bekommen als das linke. Während dies bei Festkommazahlen einen BÜ-Alarm gibt, wird bei Gleitkommazahlen dieser Überlauf durch Denormalisieren beseitigt; es wird also um 4 Stellen nach rechts geschiftet und der Exponent um 1 erhöht.

Wird der Exponent der Gleitkommazahl kleiner als -127, so sprechen wir vom Exponenten-Unterlauf. In diesem Falle wird normalisierte Gleitkomma-Null

$$-0 \cdot 16^{-127}$$

erzeugt. Eine Alarmmeldung erfolgt nicht.

Wird der Exponent größer als +127, so sprechen wir vom Exponenten-Überlauf. In diesem Falle wird ein BÜ-Alarm gegeben. Der Exponent bleibt in der übergelaufenen Form stehen, d. h., er ist um den Wert 255 zu klein. Die Mantisse ist richtig.

Ein BÜ-Alarm führt nicht direkt zur Unterbrechung. Diese Meldung wird vorerst nur vom Rechenwerk festgehalten.

Sind Über- oder Unterläufe zu erwarten, so ist es zweckmäßig, mit dem Befehl SAA - Springe, wenn arithmetischer Alarm (BÜ-Alarm) - eine Abfrage nach dem Über- oder Unterlauf vorzusehen. Im Alarmfall kann dann an

eine andere Stelle des Programms gesprungen werden, um dort das Entsprechende zu veranlassen. Gleichzeitig wird die Alarmmeldung gelöscht. Soll der Über- oder Unterlauf ignoriert werden, so besteht die Möglichkeit, mit dem Befehl SAA auf den nächsten Befehl zu springen. Der Befehl SAA hat dann nur das Löschen des BÜ-Alarmes zur Folge.

Wird ein BÜ-Alarm nicht gelöscht, so können nur die Befehle, die das Befehlswerk benutzen, ausgeführt werden. Erst in dem Augenblick, in dem ein Befehl das Rechenwerk benutzen will, erfolgt (vor der Ausführung des Befehls) ein Eingriff mit Fehlermeldung "BÜ-Alarm". Weiteres siehe Abschnitt Fehlerbehandlung.

2. 7. Rundung

Für die Zwischenergebnisse wird die Mantisse verlängert. Das Ergebnis wird dann normalisiert und danach gerundet. Waren die Operanden nicht normalisiert, so ist es möglich, daß das Ergebnis nicht gerundet ist.

Die Rundung von Gleitkommazahlen erfolgt wie bei Dezimalzahlen. Gerundet wird nur die Mantisse.

Ist die erste (linke) derjenigen Stellen, die durch Rundung wegfallen sollen, gleich oder größer 5, wird aufgerundet; im anderen Fall wird abgerundet.

$$\begin{aligned} 0,365\ 499 &= 0,365 \\ 0,365\ 500 &= 0,366 \end{aligned}$$

Beim Runden einer positiven Zahl wird folgende Operation durchgeführt:

Ist von den fortfallenden Stellen die linke Stelle L, so wird die gerundete Zahl um L erhöht.

Abrunden:	Aufunden:
$\infty, L0L0 \quad 0L \triangleq +0,640$	$\infty, L0L0 \quad L0 \triangleq +0,65625$
$\infty, L0L0 \quad \vdots$	$\infty, L0L0 \quad \vdots$
$+ \quad 0 \leftarrow$	$+ \quad L \leftarrow$
$\infty, L0L0 \quad \triangleq +0,625$	$\infty, L0LL \quad \triangleq +0,6875$

Negative Zahl: Ist von den fortfallenden Stellen die linke Stelle 0, so wird die gerundete Zahl um L erniedrigt.

Abrunden:	Aufunden:
$LL, 0L0L \quad L0 \triangleq -0,640$	$LL, 0L0L \quad 0L \triangleq -0,65625$
$LL, 0L0L \quad \vdots$	$LL, 0L0L \quad \vdots$
$- \quad 0 \leftarrow$	$- \quad L \leftarrow$
$LL, 0L0L \quad \triangleq -0,625$	$LL, 0L00 \quad \triangleq -0,6875$

Für die Rundung wird nur die linke Vorzeichenstelle abgefragt.

Würde durch eine Rundung eine über- oder untergelauene Mantisse entstehen, so wird erst um 4 Binärstellen denormalisiert und dann gerundet.

2.8. Vergleiche

Gleitkommazahlen einfacher Wortlänge können entweder mit dem Inhalt des Registers H oder aber mit Null verglichen werden. In beiden Fällen gibt es folgende Möglichkeiten:

<u>Springe wenn</u>	<u><H> ± 0</u>
identisch	SI S10
nicht identisch	SNI SN0
größer	SG SG0
größer gleich	SGG SGG0
kleiner	SK SK0
kleiner gleich	SKG SKG0

Der Vergleich erfolgt wie bei Dezimalzahlen. Das Vorzeichen wird stets berücksichtigt; positive und negative Null sind gleichwertig.

Beim Vergleich mit dem Register H werden beide Operanden normalisiert.

Des Weiteren kann mit dem Befehl SEGG abgefragt werden, ob der Exponent einen bestimmten Wert hat.

Ist eine angegebene Bedingung erfüllt, so wird ein Sprung auf den Befehl mit der Adresse, die im Adressenteil angegeben ist, ausgeführt. Vergleiche dieser Art dienen im allgemeinen zur Programmverzweigung.

Als weitere Sprungkriterien sind möglich:

- rechtes Bit vom Register A
- Markenregister
- Typenkennung
- Merklichter
- Arithmetischer Alarm
- Typenkennungsalarm

Nähere Einzelheiten zu den Sprungbefehlen sind im Abschnitt "Sonstige Operationen" erläutert.

2.9. Betrag

Der Betrag einer Gleitkommazahl wird so gebildet, wie es in der Mathematik üblich ist. Dabei ist der Betrag einer positiven Zahl die Zahl selbst, während bei einer negativen Zahl der Betrag dieser Zahl mit umgekehrtem Vorzeichen entspricht.

$$|+0,135| = +0,135$$

$$|-0,168| = +0,168$$

Intern werden bei der Betragsbildung alle Bits der Mantisse invertiert, wenn es sich um eine negative Zahl handelt. Damit ist die Zahl positiv. Der Exponent bleibt unverändert.

2.10. Typenkennung

Gleitkommazahlen einfacher Wortlänge haben die Typenkennung 0. Alle arithmetischen Befehle verlangen die Typenkennung 0. Die Vergleiche bei Sprungbefehlen hängen von der Typenkennung ab.

Bei den Gleitkommazahlen doppelter Wortlänge hat das erste Wort die gleiche Struktur wie die einfache Wortlänge, also auch die Typenkennung 0. Das zweite Wort bildet die Verlängerung der Mantisse und ist wie eine Festkommazahl aufgebaut; es hat die Typenkennung 1. Für die arithmetischen Operationen werden die Typenkennungen verlangt.

Bei den Befehlen ist angegeben, ob in Bezug auf die Typenkennung eine Voraussetzung zu erfüllen ist.

Ist diese Voraussetzung nicht erfüllt, so erfolgt die Meldung "TK-Alarm". Diese Meldung wird jedoch vom Rechenwerk vorerst nur festgehalten und führt nicht zu einer Unterbrechung.

Wenn eine falsche Typenkennung möglich ist, so wird man zweckmäßigerweise mit dem Befehl SAT eine Abfrage durchführen, ob TK-Alarm vorliegt. Im Alarmfall löscht dieser Befehl die Meldung "TK-Alarm" und springt an eine andere Stelle des Programms, um dort den TK-Alarm entsprechend abzuhandeln. Soll der TK-Alarm ignoriert werden, so kann ein Sprung auf die nächste Zeile erfolgen. Der Befehl hat dann nur die Wirkung, daß der TK-Alarm gelöscht wird.

Bleibt der TK-Alarm bestehen, so werden diejenigen Befehle, die nur das Befehlswerk benutzen, noch ausgeführt. Erst dann, wenn ein Befehl das Rechenwerk benutzen will, wird vor der Ausführung ein Eingriff mit der Fehlermeldung "TK-Alarm" gegeben. Weiteres siehe Abschnitt Fehlerbehandlung.

Sind die Bedingungen für einen TK- und einen BÜ-Alarm gegeben, dann wird der TK-Alarm stets vorrangig behandelt.

Das Ändern der Typenkennung dürfte kaum in Frage kommen.

2.11. Marke

Alle Zahlwörter können im Speicher mit einer Marke versehen werden. Ein Zahlwort bezeichnet man als markiert, wenn das erste Bit auf L gesetzt ist. Beim Transport eines markierten Zahlwortes in eines der Rechenwerksregister wird das Markenregister auf L gesetzt, sofern es zuvor auf 0 stand. Wird dagegen ein nicht markiertes Wort in ein Rechenwerksregister gebracht, so bleibt das Markenregister unverändert. Das heißt mit anderen Worten: War das Markenregister bereits vorher auf L gesetzt, dann bleibt es L, ob nun ein markiertes oder ein unmarkiertes Zahlwort ins Rechenwerk transportiert wird. Bei Operationen, in denen eine Marke ausgewertet wird, muß zweckmäßigerweise das Markenregister zu Beginn des Programms mit einem entsprechenden Befehl (z.B. LA M) gelöscht (auf 0 gesetzt) werden.

$\langle M \rangle$	$:=$	$\langle M \rangle$	V	$\langle n \rangle_n$
0		0		0
L		0		L
L		L		0
L		L		L

Dieser Vorgang ist bei der Beschreibung der Wirkung der Befehle durch die Formel $\langle M \rangle := \langle M \rangle V \langle n \rangle_n$ ausgedrückt. Ist diese Formel nicht angegeben, wird das Wort unverändert übertragen, und der vorstehend beschriebene Vorgang findet nicht statt.

Bei doppelt langen Wörtern bewirken die Befehle BZ und BZN, daß das Markenregister auf L gesetzt wird, wenn eines der Wörter oder beide markiert sind ($\langle M \rangle := \langle M \rangle V \langle n \rangle_n V \langle n+2 \rangle_n$).

Gleichzeitig mit dem Transport des Wortes und dem evtl. Setzen des Markenregisters wird das erste Bit des Zahlwortes dem zweiten Bit angeglichen. Damit sind die beiden Vorzeichenbits gleich.

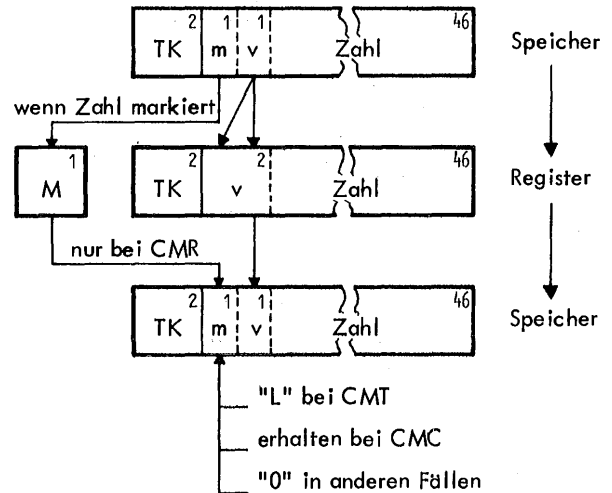
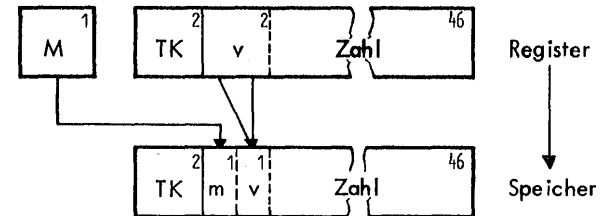


Bild 2.5 Transport der Marke

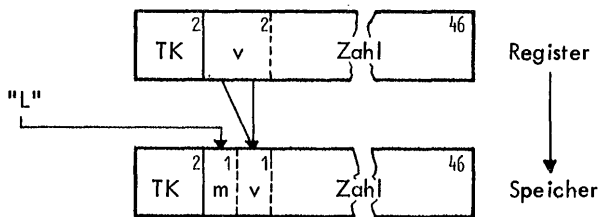
Im allgemeinen wird beim Abspeichern eines Zahlwortes aus dem Register das erste Bit auf Null gesetzt, das Wort wird also nicht markiert.

Folgende Befehle bilden eine Ausnahme:

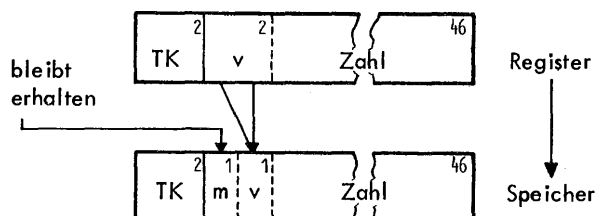
CMR Der Inhalt des Markenregisters wird in das erste Bit abgespeichert. *



CMT Das Zahlwort wird beim Abspeichern markiert (1. Bit auf "L" gesetzt). *

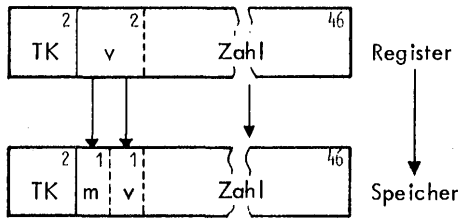


CMC Das 1. Bit in der Speicherzelle wird nicht verändert. Stand also in der Speicherzelle ein markiertes Wort, so ist auch das neue markiert. *



* Beide Vorzeichenbits des im Register stehenden Zahlwortes müssen gleich sein.

CU, C2, QCR Diese Befehle speichern unverändert, d.h. so wie im Register das entsprechende Bit gesetzt ist, wird auch das 1. Bit im Speicher gesetzt.



2.11.1. Setzen der Marke

Im Speicher kann ein Zahlwort markiert werden durch:

ZMC	n	Setze Marke im Speicher	$\langle n \rangle_n := L$
-----	---	-------------------------	----------------------------

In der Speicherzelle n wird das Markenbit $\langle n \rangle_n$ (1. Bit) auf L gesetzt.

GMT	n	Speichere markiert	$\langle n \rangle_n := L$ $\langle n \rangle := \langle A \rangle$
-----	---	--------------------	--

In der Speicherzelle n wird beim Abspeichern des Registers A das Markenbit auf L gesetzt.

LMT	n	Lösche markiert	$\langle n \rangle_n := +0$ $\langle n \rangle_n := L$
-----	---	-----------------	---

Die Speicherzelle n wird gelöscht und das Markenbit auf L gesetzt.

Das Markenregister M kann gesetzt werden durch:

ZTR	M	Setze Typenkennung im Register	$\langle M \rangle := L$
-----	---	--------------------------------	--------------------------

Ist bei dem Befehl ZTR als Spezifikation M angegeben, so wird das Register M auf L gesetzt.

Danach besteht die Möglichkeit, mit dem Befehl CMR das Markenregister abzuspeichern.

2.11.2. Löschen der Marke

Im Speicher kann die Marke gelöscht werden durch:

LMC	n	Lösche Marke im Speicher	$\langle n \rangle_n := 0$
-----	---	--------------------------	----------------------------

Das Markenbit in der Speicherzelle n wird auf 0 gelöscht.

LC	n	Lösche Speicher	$\langle n \rangle := +0$ $\langle n \rangle_n := \langle n \rangle_n$
----	---	-----------------	---

Mit dem Befehl LC kann die Speicherzelle n gelöscht werden, während das Markenbit erhalten bleibt.

Das Markenregister kann gelöscht werden durch:

LA	M	Lösche in A	$\langle M \rangle := 0$
----	---	-------------	--------------------------

Ist bei dem Befehl LA die Spezifikation M angegeben, so wird das Markenregister auf 0 gelöscht.

2.11.3. Abfragen der Marke

Das Markenregister kann durch zwei bestimmte Sprungbefehle abgefragt werden. In Abhängigkeit von einem entsprechenden Inhalt des Markenregisters ist es mit Hilfe eines Sprunges möglich, eine Verzweigung im Programm durchzuführen.

SM	m	Sprunge wenn Marke	$\langle M \rangle = L$
----	---	--------------------	-------------------------

Ist die Sprungbedingung erfüllt, wird das Markenregister auf 0 gelöscht.

SMN	m	Sprunge wenn Marke nicht	$\langle M \rangle = 0$
-----	---	--------------------------	-------------------------

Ist die Sprungbedingung nicht erfüllt, wird das Markenregister auf 0 gelöscht.

Beide Sprungbefehle hinterlassen in jedem Fall ein auf 0 gelöschttes Markenregister.

K

2.12. Einerrücklauf

Der Einerrücklauf tritt auf, wenn

- a) die Operanden verschiedene Vorzeichen haben und das Ergebnis positiv ist

$$[+] + [-] \rightarrow [+]$$

$$[-] - [+] \rightarrow [+]$$

- b) die Operanden beide negativ sind

$$[-] + [-] \rightarrow [-]$$

Dabei entsteht in der links nicht mehr vorhandenen Stelle des Ergebniswortes (also links von der höchsten vorhandenen Stelle) ein L. Es wird zur rechten Stelle des Ergebniswortes (kleinste Stelle, also ganz rechts) hinzuaddiert. Die Zahl hat dann ihren richtigen Wert. Der Einerrücklauf tritt in der externen Darstellung nicht in Erscheinung.

Bei Gleitkommazahlen muß die Mantisse und der Exponent getrennt betrachtet werden.

Beispiele:

Die Subtraktion erfolgt in Form einer Addition. Aus diesem Grund wird intern der zweite Operand invertiert und damit negativ. Anschließend wird dann addiert.
 $a-b=a+(-b)$

a) $(+10) - (+3) \triangleq (+10) + (-3)$

$$\begin{array}{r} 00\text{LOLO} \triangleq +10 \\ \underline{11\text{LL00} \triangleq -3} \\ 11\text{OLLO} \\ \quad \text{L} \\ \hline 00\text{OLLL} \triangleq +7 \end{array}$$

b) $-(+10) - (+3) \triangleq +(-10) + (-3)$

$$\begin{array}{r} 11\text{OLOL} \triangleq -10 \\ \underline{11\text{LL00} \triangleq -3} \\ 11\text{000L} \\ \quad \text{L} \\ \hline 11\text{00LO} \triangleq -13 \end{array}$$

3. OPERATIONEN FÜR EINFACHE WORTLÄNGE

Nachstehend sind die Operationen für Gleitkommazahlen einfacher Wortlänge dargestellt.

Beim Multiplikationsbefehl MLD werden zwei Operanden einfacher Wortlänge zu einem Produkt doppelter Wortlänge verknüpft. Dieser Befehl ist in diesem Abschnitt aufgeführt.

3.1. Bringen

Zur Verarbeitung werden die Gleitkommazahlen mit Hilfe der Bringebefehle in die Register A, Q, D oder H des Rechenwerkes gebracht. Dabei sind bei den Bringebefehlen nur geradzahlige Adressen zugelassen; ungeradzahlige Adressen werden stets um 1 vermindert.

Bei Gleitkommazahlen (Typenkennung 0) wird mit den meisten Bringebefehlen die Marke berücksichtigt. Diese Wirkung wird bei der Beschreibung der Befehle im Befehlslexikon durch die Gleichung

$$\langle x \rangle_1 := \langle x \rangle_B$$

$$\langle M \rangle := \langle M \rangle \vee \langle n \rangle_n$$

x: Register A, Q, D oder H

zum Ausdruck gebracht und bei der "Großen Befehlsliste" durch das Zeichen x in der Spalte $\langle M \rangle$. Alle Befehle, bei denen diese Gleichung angegeben ist, setzen bei einem markierten Zahlwort (1. Bit = L) das Markenregister auf L, unabhängig davon, ob es vorher gesetzt (L) oder nicht gesetzt (0) war. Während des Transportes wird dann das erste Bit dem zweiten Bit (Vorzeichenbit) angeglichen, so daß im Register das Zahlwort 2 Vorzeichenstellen hat, die gleich sind. Im Speicher bleibt die Zahl unverändert. Bei dem Befehl BU und BL wird die Marke nicht berücksichtigt.

Mit dem Befehl RT können die Inhalte zweier Register gegeneinander ausgetauscht werden.

3.1.1. Bringen nach Register A

Für das Bringen einer Gleitkommazahl einfacher Wortlänge aus dem Kernspeicher in das Register A sind die folgenden Befehle vorgesehen:

B	n	Bringe	$\langle A \rangle := \langle n \rangle$
BN	n	Bringe negativ	$\langle A \rangle := -\langle n \rangle$
BB	n	Bringe Betrag	$\langle A \rangle := \langle n \rangle $

Ist die Gleitkommazahl im Speicher markiert, so wird das Markenregister gesetzt.

Mit dem Befehl BN wird die Gleitkommazahl aus der Speicherzelle n mit umgekehrtem Vorzeichen ins Register A gebracht. Der Befehl BB veranlaßt den Transport des Betrages einer Gleitkommazahl aus dem Speicher in Register A. Ist die Gleitkommazahl positiv, so ist ihr Betrag identisch; ist sie negativ, so wird das Vorzeichen umgekehrt. Damit ist die Zahl positiv.

Die übrigen Rechenwerksregister werden durch diese Befehle nicht verändert.

3.1.2. Bringen in andere Register

Außer in das Register A kann eine Gleitkommazahl einfacher Wortlänge aus dem Speicher in eines der übrigen Rechenwerksregister Q, D oder H gebracht werden. Dazu stehen folgende Befehle zur Verfügung:

BQ	n	Bringe nach Q	$\langle Q \rangle := \langle n \rangle$
BD	n	Bringe nach D	$\langle D \rangle := \langle n \rangle$
BH	n	Bringe nach H	$\langle H \rangle := \langle n \rangle$
BQB	n	Bringe nach Q und bringe (nach A)	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n \rangle$

Mit dem Befehl BQB wird eine Gleitkommazahl aus der Speicherzelle n in das Register Q und außerdem gleichzeitig in das Register A gebracht.

Ist die Gleitkommazahl im Speicher markiert, so wird das Markenregister gesetzt.

3.1.3. Bringen in Register und reservieren

Bisweilen ist es nötig, daß vor dem Transport aus dem Speicher in das Register A der alte Registerinhalt für spätere Operationen reserviert wird. Diese Möglichkeit ist mit den Befehlen

BR	n	Bringe und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := \langle n \rangle$
BNR	n	Bringe negativ und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := -\langle n \rangle$

K

IAS - Handbuch

Dez. 68

gegeben. Erst dann, wenn der alte Inhalt der Registers A im Register H sichergestellt ist, wird die Gleitkommazahl einfacher Wortlänge aus der Speicherzelle n in das Register A gebracht. (Mit dem Befehl CR kann die Reserve wieder zurückgeholt werden.) Die Typenkennung des zu reservierenden Wortes ist beliebig.

Bei dem Befehl BNR wird die Zahl mit umgekehrtem Vorzeichen (d.h. invertiert) aus dem Speicher ins Register A gebracht.

War die Gleitkommazahl im Speicher markiert, so wird das Markenregister gesetzt.

3.1.4. Bringen und speichern

Um den Inhalt des Registers A mit dem Inhalt der Speicherzelle n zu vertauschen, steht der Befehl

BC	n	<u>Bringe und speichere</u>	$\langle A \rangle := \langle n \rangle$
----	---	-----------------------------	--

zur Verfügung.

War der Inhalt des Registers A und der Speicherzelle n ein Zahlwort einfacher Wortlänge, so bleibt die ursprüngliche Markenstelle in der Speicherzelle erhalten.

War die Gleitkommazahl im Speicher markiert, so wird das Markenregister gesetzt.

3.1.5. Bringen von zwei Wörtern

Zwei Gleitkommazahlen einfacher Wortlänge, die in hintereinanderliegenden Speicherzellen stehen, können mit den Befehlen

BZ	n	<u>Bringe zwei Wörter</u>	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n+2 \rangle$
BZN	n	<u>Bringe zwei Wörter negativ</u>	$\langle A \rangle := -\langle n \rangle$ $\langle Q \rangle := -\langle n+2 \rangle$

in die Register A und Q gebracht werden. Die Zahlen können dabei verschiedene Vorzeichen haben. Bei dem

Befehl BZ stehen nach dem Transport aus dem Speicher die Zahlen mit dem Vorzeichen in den Registern A und Q; bei BZN erscheinen sie dort mit umgekehrten Vorzeichen. Sind beide oder eine der zwei Zahlen markiert, so wird das Markenregister auf L gesetzt.

Der Befehl BZ n hat dieselbe Wirkung wie die zwei aufeinanderfolgenden Befehle B n, BQ n + 2.

Mit den Befehlen BZ und BZN können auch Zahlen doppelter Wortlänge in das doppelt lange Register A, Q gebracht werden (näheres siehe unter Abschnitt 4.).

3.1.6. Sonstige Bringebefehle

Um den Inhalt einer Speicherzelle unabhängig von der Typenkennung in das Register A zu bringen, steht der Befehl

BU	n	<u>Bringe unverändert</u>	$\langle A \rangle := \langle n \rangle$
----	---	---------------------------	--

zur Verfügung. Es wird ein Bitmuster gebracht, d.h. alle 48 Informationsbits werden unverändert in das Register A transportiert. (Das in 2.11. über die Marke Geschriebene bleibt unberücksichtigt.)

Mehrere Gleitkommazahlen, die im Speicher in hintereinanderliegenden Zellen stehen, können mit den Befehlen WTV und WTR in ebenfalls hintereinanderliegende Speicherzellen an eine andere Stelle im Speicher gebracht werden.

Mit einigen Tabellensuchbefehlen kann aus einer Tabelle von Gleitkommazahlen die gesuchte Zahl herausgefunden werden.

3.2. Speichern

Die Ergebnisse werden durch die Speicherbefehle in den Speicher zurückgebracht. Dabei besteht die Bedingung, daß das 1. und 2. Bit identisch sind. Dies ist bei Gleitkommazahlen immer gegeben, wenn mit normalisierten Gleitkommazahlen gearbeitet wird. Eine Ausnahme bildet der Befehl CU.

Zahlen sind im Speicher nicht markiert. Das 1. Bit in der Speicherzelle wird im allgemeinen 0 gesetzt. Eine Ausnahme stellen die Befehle CMT, CMR und CMC dar (siehe auch Abschnitt 2.11.). Durch Ihre Anwendung kann im Speicher eine Zahl markiert werden.

- CMT** das Zahlwort wird markiert $\langle n \rangle_n := L$
CMR der Inhalt des Registers M wird abgespeichert $\langle n \rangle_n := \langle M \rangle$
CMC die Markenstelle im Speicher bleibt erhalten $\langle n \rangle_n := \langle n \rangle_n$

In allen anderen Fällen ist das abgespeicherte Zahlwort nicht markiert.

Übersicht über die im folgenden näher erläuterten Speicherbefehle:

- 3.2.1. Speichern aus Register A
- 3.2.2. Speichern aus anderen Registern
- 3.2.3. Speichern aus Register A
- 3.2.4. Speichern und bringen
- 3.2.5. Speichern von zwei Wörtern
- 3.2.6. Speichern mit Marke
- 3.2.7. Sonstige Speicherbefehle

3.2.2. Speichern aus anderen Registern

Für die Abspeicherung einer Gleitkommazahl einfacher Wortlänge aus einem der Register Q, D oder H stehen die Befehle

CQ	n	Speichere aus <u>Q</u>	$\langle n \rangle := \langle Q \rangle$
CD	n	Speichere aus <u>D</u>	$\langle n \rangle := \langle D \rangle$
CH	n	Speichere aus <u>H</u>	$\langle n \rangle := \langle H \rangle$

zur Verfügung.

Bei Anwendung dieser Befehle sind die Zahlen im Speicher nicht markiert.

3.2.3. Speichern aus Register A und Reserve bringen

Mit dem Befehl

CR	n	Speichere und bringe <u>Reserve</u>	$\langle n \rangle := \langle A \rangle$ $\langle A \rangle := \langle H \rangle$
----	---	-------------------------------------	--

kann eine Gleitkommazahl einfacher Wortlänge aus dem Register A in den Speicher zurückgebracht werden. Außerdem wird anschließend der Inhalt des Registers H in das Register A gebracht. Der Befehl CR kann z.B. im Zusammenhang mit dem Befehl BR bzw. BNR verwendet werden. Bei Anwendung des Befehls CR sind die Zahlen im Speicher nicht markiert.

3.2.1. Speichern aus Register A

Eine Gleitkommazahl einfacher Wortlänge, die im Register A steht, kann mit den Befehlen

C	n	Speichere	$\langle n \rangle := \langle A \rangle$
CN	n	Speichere <u>negativ</u>	$\langle n \rangle := -\langle A \rangle$
CB	n	Speichere <u>Betrag</u>	$\langle n \rangle := \langle A \rangle $

in den Speicher zurückgebracht werden.

Der Befehl CN speichert die Zahl mit ihrem umgekehrten Vorzeichen ab. Mit dem Befehl CB wird der Betrag der Gleitkommazahl aus dem Register A zurückgespeichert. Ist die Zahl positiv, so ist ihr Betrag identisch mit der Zahl; ist sie negativ, so wird die Zahl vor dem Abspeichern invertiert; damit ist die Zahl positiv (siehe Abschnitt 2.9.).

Bei Anwendung der Befehle C, CN und CB sind die Zahlen im Speicher nicht markiert.

3.2.4. Speichern und bringen

Das Vertauschen zweier Gleitkommazahlen einfacher Wortlänge, die im Register A und in der Speicherzelle n stehen, ist mit dem Befehl

BC	n	<u>Bringe und Speichere</u>	$\langle n \rangle := \langle A \rangle$
----	---	-----------------------------	--

möglich. War die Zahl im Speicher markiert, so bleibt das Markenbit dort erhalten und das Markenregister wird gesetzt.

Die Ausführung ist die gleiche, wie wenn die Befehle B n und CMC n gleichzeitig angewendet würden.

K

TAS - Handbuch

Dez. 68

3.2.5. Speichern von zwei Wörtern

Mit Hilfe des Befehls

CZ	n	Speichere zwei Wörter	$\langle n \rangle := \langle A \rangle$ $\langle n+2 \rangle := \langle Q \rangle$
----	---	-----------------------	--

können zwei Gleitkommazahlen einfacher Wortlänge aus den Registern A und Q in zwei hintereinanderliegende Speicherzellen zurückgebracht werden. Dieser Befehl hat die gleiche Wirkung wie die beiden aufeinanderfolgenden Befehle C n, CQ n+2.

Mit CZ ist es außerdem möglich, ein doppelt langes Ergebnis aus dem doppelt langen Register AQ in den Speicher zurückzubringen (siehe Abschnitt 4.).

Bei Anwendung des Befehls sind die Zahlen im Speicher nicht markiert.

3.2.6. Speichern mit Marke

Bei den Befehlen

CMT	n	Speichere <u>markiert</u>	$\langle n \rangle := \langle A \rangle$ $\langle n \rangle_m := L$
CMR	n	Speichere mit <u>Marke</u> aus <u>Register</u>	$\langle n \rangle := \langle A \rangle$ $\langle n \rangle_m := \langle M \rangle$
CMC	n	Speichere mit <u>Marke</u> aus <u>Speicher</u>	$\langle n \rangle := \langle A \rangle$ $\langle n \rangle_m := \langle n \rangle_m$

wird eine Marke berücksichtigt.

Um ein Zahlwort, das vom Register A in den Speicher gebracht worden ist, auf jeden Fall zu markieren, wird der Befehl CMT angewendet. Das erste Bit in der Speicherzelle n wird durch CMT immer auf L gesetzt.

Der Befehl CMR bewirkt, daß die Markenstelle eines Zahlwortes, das aus dem Register A abgespeichert wurde, mit dem Inhalt des Registers M belegt wird. Die Zahl im Speicher ist also dann markiert, wenn im Register M ein L steht.

Der Befehl CMC ermöglicht die Abspeicherung einer Zahl aus dem Register A, wobei das Markenbit, das in der Speicherzelle n steht, unverändert bleibt. Während also die im Speicher vorhandene Zahl durch die neue Zahl aus dem Register A überschrieben wird, bleibt die ursprüngliche Markenstelle erhalten.

3.3. Addition - Subtraktion

Zur Durchführung einer Subtraktion wird beim Subtrahend das Vorzeichen gewechselt (d.h. die Mantisse wird invertiert). Anschließend erfolgt eine Addition der beiden Operanden. Aus diesem Grund können die Beschreibungen der Addition und der Subtraktion zusammengefaßt werden.

Um eine Addition bzw. eine Subtraktion durchführen zu können, muß der eine Operand bereits in einem Register des Befehlswerkes vorhanden sein. Der andere Operand wird durch den Additions- bzw. Subtraktionsbefehl vom Speicher in das Rechenwerk gebracht. Bezüglich der Marke gilt das im Abschnitt 3.1. über den Bringebefehl Gesagte.

Die Operanden müssen stets die Typenkennung 0 haben, im anderen Fall wird TK-Alarm gegeben. Wird beim Ergebnis der Bereich überschritten (Exponent $> +127$), so wird BÜ-Alarm gegeben.

Das Register Q wird intern für die Gleitkommaoperationen benötigt und wird am Ende der Operation in allen Binärstellen, einschließlich Typenkennung auf 0 glöscht. Es steht dann also eine Gleitkommazahl mit dem Wert 0 ($+0 \cdot 16^{+0}$) im Register Q.

3.3.1. Im Register A

Für die Addition und Subtraktion im Register A stehen folgende Befehle zur Verfügung

GA	n	Gleitkomma <u>addiere</u>	$\langle A \rangle := \langle A \rangle + \langle n \rangle$
GSB	n	Gleitkomma <u>subtrahiere</u>	$\langle A \rangle := \langle A \rangle - \langle n \rangle$
GSBI	n	Gleitkomma <u>subtrahiere invers</u>	$\langle A \rangle := \langle n \rangle - \langle A \rangle$

Der eine Operand steht im Register A und der andere wird aus der Speicherzelle n geholt. Das Ergebnis steht im Register A.

Während beim Befehl GSB der Subtrahend aus dem Speicher geholt wird, wird beim Befehl GSBI der Minuend aus dem Speicher geholt.

Als Nebenwirkung bei den Befehlen steht der Operand, der aus dem Speicher geholt wurde im Register D.

3.3.2. Im Speicher

Mit den Befehlen GAC und GSBC stehen ebenfalls die Operanden im Register A und in der Speicherzelle n.

GAC	n	Gleitkomma <u>addiere</u> im Speicher	$\langle n \rangle := \langle A \rangle + \langle n \rangle$
GSBC	n	Gleitkomma <u>subtra-</u> <u>hiere</u> im Speicher	$\langle n \rangle := \langle n \rangle - \langle A \rangle$

Das Ergebnis wird jedoch sofort in der Speicherzelle n unmarkiert abgespeichert. Der Operand im Register A bleibt unverändert.

Als Nebenwirkung steht das Ergebnis gleichzeitig im Register D.

Als Nebenwirkung steht der Operand aus der Speicherzelle n im Register D.

3.3.5. Unnormalisiertes Ergebnis

Bei allen arithmetischen Gleitkommabefehlen ist das Ergebnis normalisiert, wenn auch die Operanden normalisiert waren (siehe Abschnitt 2.5.).

Eine Ausnahme bilden die Befehle

AU	n	Addiere unnormalisiert	$\langle A \rangle := \langle A \rangle + \langle n \rangle$
SBU	n	Subtrahiere unnormalisiert	$\langle A \rangle := \langle A \rangle - \langle n \rangle$

3.3.3. Im Register D

Beim Befehl GSBD befindet sich der eine Operand im Unterschied zum Befehl GSB nicht im Register A, sondern im Register D.

GSBD	n	Gleitkomma <u>subtra-</u> <u>hiere</u> von <u>D</u>	$\langle A \rangle := \langle D \rangle - \langle n \rangle$
------	---	--	--

Es wird also vom Inhalt des Registers D der Inhalt der Speicherzelle n subtrahiert. Das Ergebnis steht im Register A. Das Register D bleibt unverändert.

Sie wirken wie die Befehle GA bzw. GSB, jedoch ist das Ergebnis nicht normalisiert. Diese Befehle werden nur in sehr speziellen Fällen benötigt.

Als Nebenwirkung wird der Operand aus dem Speicher in das Register D gebracht.

3.3.4. Betrag

Mit den Befehlen GAB und GSBB kann der Betrag einer Gleitkommazahl addiert bzw. subtrahiert werden.

GAB	n	Gleitkomma <u>addiere</u> <u>Betrag</u>	$\langle A \rangle := \langle A \rangle + \langle n \rangle $
GSBB	n	Gleitkomma <u>subtrahiere</u> <u>Betrag</u>	$\langle A \rangle := \langle A \rangle - \langle n \rangle $

Der erste Operand steht im Register A. Der zweite Operand wird aus der Speicherzelle n geholt und dessen Betrag addiert bzw. subtrahiert. Das Ergebnis steht im Register A.

Im Bild 3.1. ist ein Beispiel für den Befehl AU gebracht. Die Gleitkommazahl in der Speicherzelle GKZAHL sei kleiner als 2^{36} . Mit der Befehlsfolge wird erreicht, daß der ganzzahlige Anteil als Festkommazahl dargestellt wird. Dann wird mit dem Befehl AU die Zahl $+0 \cdot 16^{19}$ addiert. Der Wert der Gleitkommazahl wird damit nicht verändert. Auf Grund der Bedingung, daß bei der Addition die Exponenten identisch sein müssen, wird die Gleitkommazahl denormalisiert, bis der Exponent den Wert $16^9 = 2^{36}$ hat. Der Exponent zeigt an, daß das Komma, das bei der Mantisse rechts von 2. steht, tatsächlich rechts von der 38. Stelle ($2 + 4 \cdot 9$) steht. Das Ergebnis wird nicht normalisiert, die Gleitkommazahl bleibt also denormalisiert stehen. Wird nun das Bitmuster der Gleitkommazahl um 10 Stellen nach rechts geschiftet, so wird der Exponent herausgeschiftet und die Mantisse um 10 Stellen nach rechts geschiftet. Das Komma der Mantisse steht dann ganz rechts hinten an der 48. Stelle. Damit haben wir eine ganze Festkommazahl, bei der noch die Typenkennung auf 1 zu setzen ist.

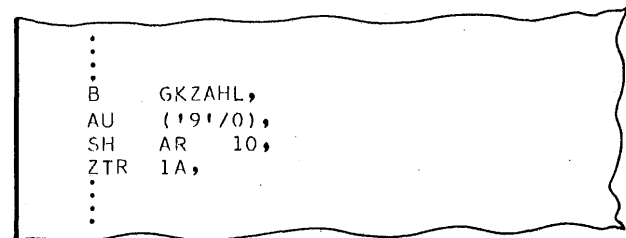


Bild 3.1 Beispiel zum Befehl AU

K

TAS - Handbuch

Dez. 68

3.3.6. Sonstiges

Bei den Befehlen GMLA und GMAN wird nach einer Multiplikation der Inhalt des Registers D zum Ergebnis addiert. (Siehe dazu Abschnitt 3.4.2.).

Bei den Befehlen AA und SBA wird der Adressenteil z zum Exponenten addiert bzw. von ihm subtrahiert. Dies bedeutet jedoch, daß die Gleitkommazahl mit dem Faktor 16^z multipliziert bzw. dividiert wird. Siehe dazu Abschnitt 3.4.3.

3.4. Multiplikation

Werden zwei Gleitkommazahlen einfacher Länge multipliziert, so ist das Ergebnis wiederum eine Gleitkommazahl einfacher Länge. Darüber hinaus kann mit dem Befehl MLD auch als Ergebnis eine Gleitkommazahl doppelter Länge entstehen.

Werden normalisierte Operanden verwendet, so ist maximal eine Normalisierung von einmal vier Stellen notwendig und daher auch nur ausgeführt. Werden also nicht-normalisierte Operanden verwendet, so ist es möglich, daß auch das Ergebnis nicht normalisiert ist.

Die Typenkennung der Operanden muß stets 0 sein. Im anderen Fall wird TK-Alarm gegeben.

Wird beim Ergebnis der zulässige Bereich überschritten (Exponent $> +127$), so wird BÜ-Alarm gegeben.

Für die Multiplikation wird intern das Register Q benötigt. Es wird am Ende der Multiplikation in allen Binärstellen einschließlich Typenkennung auf 0 gelöscht. Es steht an dann also eine Gleitkommazahl mit dem Wert $0 (+0 \cdot 16^{+0})$ im Register Q.

Als Nebenwirkung wird der Operand, der aus der Speicherzelle n geholt wird, in das Register D gebracht.

3.4.1. Einfach

Für die einfache Multiplikation stehen die Befehle GML und GMLN zur Verfügung

GML	n	Gleitkomma multipliziere	$\langle A \rangle := \langle A \rangle \cdot \langle n \rangle$
GMLN	n	Gleitkomma multipliziere negativ	$\langle A \rangle := -\langle A \rangle \cdot \langle n \rangle$

Der eine Operand muß im Register A stehen, der andere Operand wird aus der Speicherzelle n geholt. Das Ergebnis steht im Register A. Beim Befehl GMLN wird außerdem beim Ergebnis das Vorzeichen gewechselt.

3.4.2. Mit Addition

Eine Erweiterung der vorstehenden Befehle bilden die Befehle GMLA und GMAN

GMLA	n	Gleitkomma multipl. akkumulierend	$\langle A \rangle := \langle A \rangle \cdot \langle n \rangle + \langle H \rangle$
GMAN	n	Gleitkomma multipl. akkum. negativ	$\langle A \rangle := -\langle A \rangle \cdot \langle n \rangle + \langle H \rangle$

Bei diesen Befehlen wird wie bei den entsprechenden Befehlen GML und GMLN multipliziert. Danach wird der Operand, der im Register H ist, hinzu addiert. Das Ergebnis steht wiederum im Register H.

Diese Befehle eignen sich besonders in Verbindung mit den Bringebefehlen BR und BRN, die einen Operanden, der im Register A steht, im Register H reservieren.

3.4.3. Mit Adressenteil

Bei den Befehlen AA und SBA wird der Adressenteil dieser Befehle zum Inhalt des Registers A addiert. Ist der Operand im Register A eine Gleitkommazahl, so darf die Zahl z im Adressenteil der Befehle AA und SBA nicht größer sein als $127 (2^7 - 1)$.

AA	z	Addiere Adressenteil (TK = 0)	$\langle A \rangle := \langle A \rangle \cdot 16^z$
SBA	z	Subtrahiere Adressenteil (TK = 0)	$\langle A \rangle := \langle A \rangle \cdot 16^{-z}$

Es werden nur die rechten 8 Bits des Adressenteils auf den Exponenten der Gleitkommazahl addiert. Dies bedeutet für Gleitkommazahlen, daß sie multipliziert bzw. dividiert werden und zwar beim Befehl AA mit dem Faktor

$$16^z$$

multipliziert und beim Befehl SBA durch diesen Faktor dividiert.

Im Adressenteil der beiden Befehle kann auch eine negative Zahl stehen, die nicht kleiner sein darf als -127 . In diesem Falle wird beim Assemblieren eine Fehlermeldung gegeben, daß das Adressenteil negativ ist. Dies kann ignoriert werden.

Als Nebenwirkung wird die im Adressenteil der Befehle stehende Zahl als Festkommazahl ins Register D gebracht. Dabei werden jedoch die Bits 1 bis 40 auf den Wert 0 gesetzt, so daß nur positive Zahlen vorzeichenrichtig im Register D stehen.

Ein Typenkennungsalarm entfällt hier, da die Typenkennung des Registers A bestimmt, wie die Befehle AA und SBA ablaufen und der Adressenteil keine Typenkennung hat.

3.4.4. Ergebnis mit doppelter Wortlänge

Mit dem Befehl MLD werden zwei Gleitkommazahlen einfacher Länge multipliziert. Das Ergebnis ist eine Gleitkommazahl doppelter Wortlänge.

MLD	n	Gleitkomma <u>multipl.</u> auf <u>doppelte Genauigkeit</u>	$\langle A, Q \rangle := \langle A \rangle \cdot \langle n \rangle$
-----	---	--	---

Der eine Operand steht im Register A, der andere wird aus der Speicherzelle n geholt. Das Ergebnis der Multiplikation steht als doppelt lange Gleitkommazahl in den Registern A und Q.

Das Ergebnis kann als doppelt lange Gleitkommazahl, z.B. mit den Befehlen DA, DSB und DML weiterverarbeitet werden (s. Abschnitt 4).

Als Nebenwirkung werden bei diesem Befehl die Register D und H mit Typenkennung 1 auf Null gelöscht.

3.5. Division

Für die Division stehen folgende Befehle zur Verfügung

GDV	n	Gleitkomma <u>dividiere</u>	$\langle A \rangle := \langle A \rangle : \langle n \rangle$
GDVI	n	Gleitkomma <u>dividiere</u> <u>invers</u>	$\langle A \rangle := \langle A \rangle : \langle n \rangle$
REZ	n	Bilde <u>reziproken</u> Wert	$\langle A \rangle := 1 : \langle n \rangle$

Sie unterscheiden sich dadurch, daß ihre Operanden verschieden sind.

Befehl	Dividend	: Divisor
GDV	Register A	Speicherzelle n
GDVI	Speicherzelle n	Register A
REZ	1	Speicherzelle n

Das Ergebnis steht in allen drei Fällen im Register A.

Soll der Reziprokwert nicht von einem Operanden aus dem Speicher gebildet werden, sondern von einem Operanden in einem der Rechenwerksregister, so ist das mit Hilfe des Befehls R möglich.

R REZ s s: Register A, Q, D oder H

Für die Operationen werden die Register Q und D benötigt. Sie werden am Schluß der Operationen als Nebenwirkung in allen Binärstellen auf den Wert 0 gesetzt. Das ist die Gleitkommazahl $+0 \cdot 16^{+0}$.

Die Operanden müssen die Typenkennung 0 haben und normalisiert sein. Wird beim Ergebnis der Exponent größer als +127, so wird BÜ-Alarm gegeben.

K

TAS - Handbuch

Mei 70

4. OPERATIONEN FÜR DOPPELTE WORTLÄNGE

Für Gleitkommazahlen doppelter Länge werden zwei Ganzwörter benötigt. Das erste Wort - es steht im Rechenwerk im allgemeinen im Register A - hat den gleichen Aufbau wie eine Gleitkommazahl einfacher Wortlänge. Das zweite Wort - es steht im Rechenwerk im allgemeinen im Register Q - ist eine Verlängerung der Mantisse des ersten Wortes und hat die gleiche Struktur wie Festkommazahlen.

Beide Wörter werden für die Befehle DA, DSB und DML als Einheit betrachtet. Adressiert wird dabei das erste Wort.

4.1. Bringen

Da eine Gleitkommazahl doppelter Wortlänge aus zwei Ganzwörtern besteht, kommen für den Transport aus dem Speicher in das Rechenwerk nur die Bringebefehle BZ und BZN in Frage.

BZ	n	Bringe zwei Wörter	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n+2 \rangle$
BZN	n	Bringe zwei Wörter negativ	$\langle A \rangle := -\langle n \rangle$ $\langle Q \rangle := -\langle n+2 \rangle$

Das Wort aus der Speicherzelle n wird in das Register A gebracht und das Wort aus der folgenden Speicherzelle (n + 2) in das Register Q. Bei dem Befehl BZN wird dabei das Vorzeichen der (doppelt langen) Gleitkommazahl gewechselt.

Bei Gleitkommazahlen mit doppelter Wortlänge, mit Typenkennung = 0 im ersten Wort und Typenkennung = 1 im zweiten Wort, wird (wie bei Festkommazahlen) das ganze Wort invertiert. Damit bringt der Befehl BZN auch Gleitkommazahlen doppelter Genauigkeit vorzeichenrichtig.

Ist eines der beiden Wörter markiert, so wird das Markenregister gesetzt. Es können hier auch die gleichen Bringebefehle wie bei Gleitkommazahlen einfacher Genauigkeit verwendet werden. Dann sind zwei Befehle erforderlich, z.B.

B n
BQ n + 2

4.2. Speichern

Für den Transport einer doppelt langen Gleitkommazahl steht der Speicherbefehl CZ zur Verfügung.

CZ	n	Speichere zwei Wörter	$\langle n \rangle := \langle A \rangle$ $\langle n+2 \rangle := \langle Q \rangle$
----	---	-----------------------	--

Er bringt die Gleitkommazahl in zwei aufeinanderfolgende Speicherzellen (n und n + 2). Dabei sind beide Wörter unmarkiert.

Darüber hinaus können die gleichen Speicherbefehle wie bei Gleitkommazahlen einfacher Genauigkeit verwendet werden; es sind dann lediglich zwei Befehle erforderlich, z.B.

C n
CQ n + 2

Dies kann dann von Interesse sein, wenn die Gleitkommazahl markiert werden soll. So wird z.B. mit der Befehlsfolge

CMT n
CQ n + 2

das erste Wort markiert. Es kann jedoch auch mit dem Befehl ZMC die Marke nachträglich gesetzt werden. So wird mit der Befehlsfolge

CZ n
ZMC n + 2

das zweite Wort markiert.

4.3. Addition - Subtraktion

Für die Addition steht der Befehl DA und für die Subtraktion der Befehl DSB zur Verfügung.

DA	n	Doppelte Genauigkeit: Addiere	$\langle A, Q \rangle := \langle A, Q \rangle + \langle n, n+2 \rangle$
DSB	n	Doppelte Genauigkeit: Subtrahiere	$\langle A, Q \rangle := \langle A, Q \rangle - \langle n, n+2 \rangle$

In beiden Fällen steht der eine Operand in den Registern A und Q und der andere Operand in den Speicherzellen n und n + 2. Das Ergebnis steht wiederum in den Registern A und Q.

Als Nebenwirkung werden durch die beiden Befehle die Register D und H mit Typenkennung 1 auf Null gelöscht.

4.4. Multiplikation

Mit dem Befehl DML können zwei Gleitkommazahlen doppelter Wortlänge multipliziert werden. Das Ergebnis ist eine Gleitkommazahl mit doppelter Wortlänge.

DML	n	Doppelte Genauigkeit: Multipliziere	$\langle A, Q \rangle := \langle A, Q \rangle \cdot \langle n, n+2 \rangle$
-----	---	-------------------------------------	---

Der eine Operand steht in den Registern A und Q, der andere in den Speicherzellen n und n + 2. Das Ergebnis der Multiplikation steht als doppelt lange Gleitkommazahl in den Registern A und Q.

Als Nebenwirkung werden bei diesem Befehl die Register D und H mit Typenkennung 1 auf Null gelöscht.

K

TAS - Handbuch

Mei 70

TELEFUNKEN
COMPUTER

TAS - HANDBUCH

Nichtzahlwörter

L

INHALT

NICHTWAHLWÖRTER	0 - 1
-----------------------	-------

<u>1. EXTERNER AUFBAU</u>	1 - 1
<u>2. INTERNER AUFBAU</u>	2 - 1
2.1. Typenkennung	2 - 1
2.2. Invertieren	2 - 1
2.3. Vergleiche	2 - 2
<u>3. SETZEN - LÖSCHEN</u>	3 - 1
3.1. Register	3 - 1
3.2. Speicher	3 - 2
3.3. Typenkennung	3 - 2
<u>4. BRINGEN</u>	4 - 1
4.1. Bringen von Ganzwörtern	4 - 1
4.2. Bringen von zwei Wörtern	4 - 2
4.3. Bringen von Halbwörtern	4 - 2
4.4. Bringen von Drittelwörtern	4 - 2
4.5. Zeichenweise bringen	4 - 2
4.6. Teilwort bringen	4 - 3
<u>5. SPEICHERN</u>	5 - 1
5.1. Speichern von Ganzwörtern	5 - 1
5.2. Speichern von zwei Wörtern	5 - 1
5.3. Speichern von Halbwörtern	5 - 1
5.4. Speichern von Drittelwörtern	5 - 2
5.5. Zeichenweise speichern	5 - 2
5.6. Teilwort speichern	5 - 3
<u>6. ABFRAGEN - VERZWEIGUNGEN - SPRÜNGE</u>	6 - 1
6.1. Vergleich Register A mit Null	6 - 1
6.2. Vergleich Register A und H	6 - 1
6.3. Abfrage eines Bits	6 - 2
6.4. Abfrage der Typenkennung	6 - 2
<u>7. SCHIFTEN</u>	7 - 1
7.1. Register	7 - 1
7.2. Schiffrichtung	7 - 1
7.3. Kreisschift	7 - 1
7.4. Typenkennung	7 - 1
7.5. Zählen der besetzten Bits	7 - 2
7.6. Beispiele	7 - 2
<u>8. ADDITION - SUBTRAKTION</u>	8 - 1
8.1. Ganzwörter	8 - 1
8.2. Halbwörter	8 - 2
8.3. Adressenteil	8 - 2



<u>9.</u>	<u>ZEICHENVERARBEITUNG</u>	9 - 1
9.1.	Bringen und Speichern	9 - 1
9.2.	Umschlüsseln	9 - 4
9.2.1.	Einzelzeichen	9 - 4
9.2.2.	Alle Zeichen	9 - 5
9.2.3.	Umschlüsseltabelle	9 - 5
9.3.	Beispiel	9 - 6
<u>10.</u>	<u>TEILWÖRTER</u>	10 - 1
10.1.	Maske	10 - 1
10.2.	Teilwort bringen	10 - 1
10.3.	Addition - Subtraktion	10 - 2
10.4.	Teilwort speichern	10 - 3
10.5.	Beispiele	10 - 3

L

NICHTZAHLWÖRTER

Durch die Typenkennung (TK) werden vier Wortarten unterschieden.

TK = 0	Gleitkommazahlen	} Zahlwörter
TK = 1	Festkommazahlen	
TK = 2	Befehle	} Nichtzahlwörter
TK = 3	Alphatext	

Dieser Abschnitt befaßt sich mit den Operationen für Wörter mit der Typenkennung 3, die also Alphatext enthalten.

Da die weitaus meisten Befehle Wörter mit der Typenkennung 2 und 3 auf die gleiche Art verarbeiten, ist dieser Abschnitt mit "Nichtzahlwörter" überschrieben. Die Ausnahmen, bei denen die Wörter mit Typenkennung 2 anders verarbeitet wurden, sind angegeben.

Bei einigen Befehlen ist es auch möglich, Alphatexte zu verarbeiten, die eine beliebige Typenkennung haben. In wichtigen Fällen ist darauf besonders eingegangen. Genaue Angaben dazu sind dem "TR 440 Befehls-Lexikon" zu entnehmen.

Unter Alphatext verstehen wir, daß die Information dieses Wortes ein beliebiges Bitmuster und auch in beliebige Bitgruppen unterteilt sein kann. Dies ermöglicht eine universelle Anwendung des Rechners. Diese Tatsache kann jedoch im Rahmen dieser Schrift nicht erschöpfend behandelt werden. Deshalb wird der am häufigsten auftretende Fall beschrieben, wo der Alphatext in Oktaden vorliegt, die dem Zentralcode entsprechen.



I. EXTERNER AUFBAU

Die zu verarbeitenden Daten werden im allgemeinen in Lochkarten oder Lochstreifen abgelocht und über die entsprechenden Eingabegeräte eingelesen. Des weiteren können sie auch über Fernschreiber, Schreibmaschine oder andere Eingabegeräte eingegeben werden.

Die Eingabe der Daten wird von speziellen Programmen, den Eingabevermittlern, durchgeführt. Sie lesen die Daten ein und übergeben sie dem Empfänger (Empfängerprogramm). Je nach Steuerung des Eingabevermittlers stehen die Daten dann in den folgenden drei Formen zur Verfügung:

- im Zentralcode (Oktaden)
- in Tetraden (nur bei Zahlen möglich)
- im Binärmuster des Eingabemediums.

Nähere Einzelheiten hierzu sind der Beschreibung des jeweiligen Eingabevermittlers zu entnehmen.

Eine weitere Möglichkeit, Daten in den Rechner einzuschleusen, besteht darin, sie im Quellenprogramm niederzuschreiben. Dem Benutzer stehen dabei die Konstanten der TAS-Sprache

- Tetradenkonstanten
- Oktadenkonstanten
- Textkonstanten
- Bitfeldkonstanten

zur Verfügung. Dies hat für die Eingabe von Daten in den Rechner jedoch nur begrenzte Bedeutung.



2. INTERNER AUFBAU

Wörter mit Alphatexten haben den im Bild 2.1 gezeigten allgemeinen Aufbau. Die eigentliche Information besteht aus 48 Bits, die beliebig codiert sein können. In Bild 2.2 sind einige Beispiele aufgeführt. Der häufigste Fall ist der, daß der Alphatext in Oktaden im Zentralcode vorliegt. Liegen im Alphatext nur Zahlen vor, so können sie in Tetraden dargestellt werden. Das letzte Beispiel zeigt die Eingabe von Lochkarten binär. Jedes Wort kann dann 4 Lochkartenspalten aufnehmen.

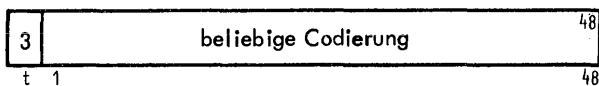
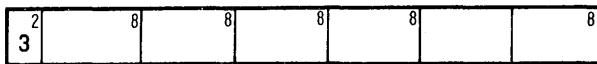
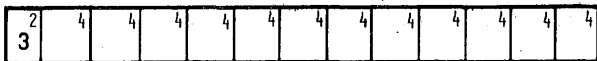


Bild 2.1 Allgemeiner Aufbau eines Alphatextes

Oktaden im Zentralcode



Tetraden



Lochkartenspalten binär

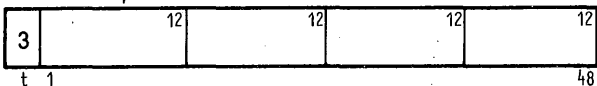


Bild 2.2 Beispiele für interne Darstellung bei Alphatexten

2.1. Typenkennung

Im Normalfall haben Alphatexte die Typenkennung 3.

Bei der Verarbeitung von Alphatexten ist es jedoch möglich, die Typenkennung zu verwenden, um

- zwei weitere Informationsbits zu gewinnen,
- ein Wort zu kennzeichnen,
- ein Abbruchkriterium bei Schleifen zu erhalten.

Dabei muß jedoch darauf geachtet werden, daß ein Teil der Befehle bei Typenkennung 0, 1 und 2 anders wirkt als bei Typenkennung 3. Es sind dies im besonderen die Transportbefehle, arithmetischen Befehle und Sprungbefehle.

Mit den Befehlen

ZTR	s	Setze Typenkennung im Register	$\langle s_2 \rangle_t := s_1$
ZT0	n	Setze Typenkennung 0	$\langle n \rangle_t := 0$
ZT1	n	Setze Typenkennung 1	$\langle n \rangle_t := 1$
ZT2	n	Setze Typenkennung 2	$\langle n \rangle_t := 2$
ZT3	n	Setze Typenkennung 3	$\langle n \rangle_t := 3$

s_1 : 0, 1, 2, 3 oder leer
 s_2 : A, Q, H, D oder leer
 s_3 : leer oder M

kann die Typenkennung im Register oder im Speicher auf einen anderen Wert gesetzt werden.

Soll die Typenkennung als Abbruchkriterium in Schleifen verwendet werden, so kann sie mit Hilfe der Befehle

ST	p s	Sprunge wenn Typenkennung	$\langle s_2 \rangle_t = s_1$
STN	p s	Sprunge wenn Typenkennung nicht	$\langle s_2 \rangle_t \neq s_1$

p: +0...+127
 s_1 : 0, 1, 2 oder 3 (TK)
 s_2 : A, Q, D oder H

in den Registern A, Q, D oder H abgefragt und in Abhängigkeit davon ein Sprung ausgeführt werden. So ist es z. B. möglich, eine Gruppe von Alphawörtern, die die Typenkennung 3 haben, mit einem Wort abzuschließen, das eine andere Typenkennung hat. Dies wird bei den Tabellensuchbefehlen angewandt.

2.2. Invertieren

Mit dem Befehl

IR	s	Invertiere Register	$\langle s \rangle := \neg \langle s \rangle$
----	---	---------------------	---

s: Register A, Q, D und H

kann ein Alphawort im Register invertiert werden. Dabei erhalten alle Binärstellen den umgekehrten Wert.

Mit den Befehlen

BN	n	<u>B</u> ringe <u>n</u> egativ	$\langle A \rangle := -\langle n \rangle$
BNR	n	<u>B</u> ringe <u>n</u> egativ und <u>r</u> eserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := -\langle n \rangle$

ist eine Invertierung der Alphawörter beim Bringen verbunden und bei den Befehlen

CN	n	<u>S</u> peichere <u>n</u> egativ	$\langle n \rangle := -\langle A \rangle$
----	---	-----------------------------------	---

ein Invertieren beim Abspeichern eines Wortes.

2.3. Vergleiche

Beim Vergleich wird ein Nichtzahlwort (TK = 2 und 3) stets als positive ganze Binärzahl aufgefaßt.

Beim Vergleich mit Null bedeutet das, daß es identisch Null ist, wenn alle Binärstellen 0 sind. Bei allen anderen Werten des Nichtzahlwortes ist es größer als Null.

Beim Vergleich mit dem Register H ist es nur dann identisch, wenn alle Binärstellen identisch sind, in allen anderen Fällen ist es nicht identisch.

3. SETZEN - LÖSCHEN

In vielen Fällen ist es erforderlich, eines der Rechenwerksregister oder eine Speicherzelle auf einen bestimmten Wert zu setzen. In diesem Sinn kann auch das Löschen als ein Setzen auf den Wert Null bezeichnet werden.

3.1. Register

Um das Register A auf einen bestimmten Wert zu setzen, stehen die Befehle

BA	z	Bringe Adressenteil	$\langle A \rangle := z$
BAR	z	Bringe Adressenteil und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := z$

z: 0...65535 (vor Modifiz.)

zur Verfügung. Der zu setzende Wert kann 16 Bits lang sein (und durch Modifizierung 24 Bits). Der im Adressenteil des Befehls stehende Wert wird rechtsbündig in das Register A gebracht. Der linke Teil wird mit 0 aufgefüllt. Der Wert erhält die Typenkennung 1. Bei dem Befehl BAR wird außerdem noch der Inhalt des Registers A im Register H reserviert.

Die Befehle BAN und BANR sind nur bedingt für Nichtzahlwörter verwendbar.

Soll ein anderes Register auf einen bestimmten Wert gesetzt werden oder das Register A auf einen Wert, der größer als 16 Bits ist, so kann dies in der TAS-Sprache mit Hilfe von Bringebefehlen erfolgen, die im Adressenteil ein Literal haben. In dem Literal wird der Wert als Konstante niedergeschrieben. So bewirkt z.B. der Befehl

BH (''100,00''),

daß der Wert 100,00 in Oktaden des Zentralcodes in das Register H gebracht wird und die Typenkennung 3 erhält.

Ein Löschen der Rechenwerksregister ist ebenfalls mit den vorstehend beschriebenen Befehlen möglich. Hier gibt es jedoch einen speziellen Befehl.

LR	s	Lösche Register	$\langle s_2 \rangle := +0$ $\langle s_2 \rangle_t := s_1$
----	---	-----------------	---

s_1 : 0,1,2 oder 3 (TK)
 s_2 : A,D,Q und H

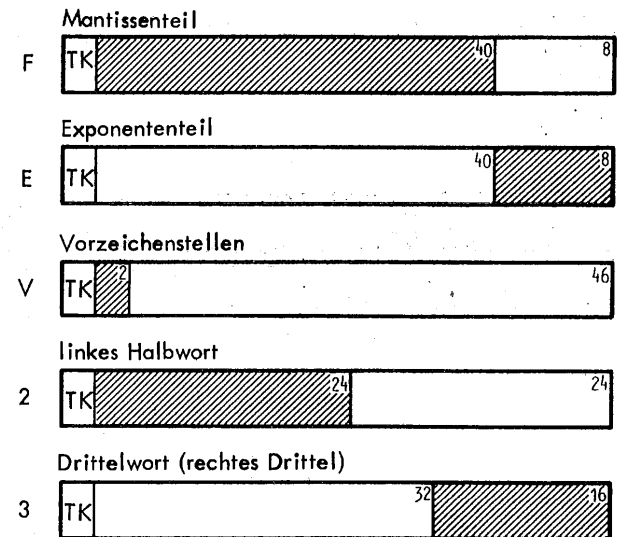
Mit ihm können die Register A, Q, D und H gelöscht werden und gleichzeitig eine bestimmte Typenkennung erhalten. Es können auch gleichzeitig mehrere oder alle Register gelöscht werden. Sie erhalten dann alle die gleiche Typenkennung.

Um im Register A nur bestimmte Teile zu löschen, steht der Befehl

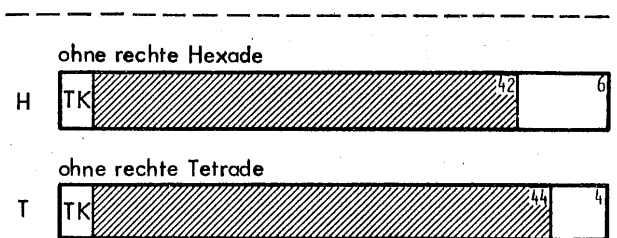
LA	s	Lösche in A	$\langle A \rangle_s := 0$ $\langle M \rangle := 0$ wenn $s = M$
----	---	-------------	---

zur Verfügung. In Bild 3.1 ist angegeben, welche Teile des Registers gelöscht werden können. Die Spezifikationen H und T dürfen nur einzeln verwendet werden oder mit der Spezifikation M zusammen. Alle anderen (einschließlich M) können kombiniert werden. Die Spezifikation F hat gleichzeitig die Bedeutung "ohne rechte Oktade". Die Spezifikation FE löscht das ganze Register.

Spezifikation



können kombiniert werden



nur einzeln oder mit M erlaubt

M Register M

kann mit allen kombiniert werden

Bild 3.1 Möglichkeiten beim Befehl LA

```

BA 100,
BA 'DM',
BAR '75',
BQ ('',-- DM'),
LR AH3,
LA E2M,

```

Bild 3.2 Beispiele für Setzen und Löschen der Register

3.2. Speicher

Eine Speicherzelle kann dadurch auf einen bestimmten Wert gesetzt werden, daß ein Register auf diesen Wert gesetzt (siehe dazu Abschnitt 3.1) und dieser Wert dann abgespeichert wird.

Mit Hilfe der TAS-Sprache können Speicherzellen auf einen Wert gesetzt werden, indem sie durch die Angabe einer Konstanten bei der Übersetzung belegt werden.

Zum Löschen einer Speicherzelle steht der Befehl

LC	n	Lösche Speicher	$\langle n \rangle := +0$ $\langle n \rangle_n := \langle n \rangle_n$
----	---	-----------------	---

zur Verfügung. Die Typenkennung der Speicherzelle bleibt erhalten. Soll die Speicherzelle mit einer bestimmten Typenkennung gelöscht werden, so ist dies z.B. über die Befehle LR und C möglich.

```

BA 118,
ZTR A3,
C ANTON,

B ('',-- DM'),
C ANTON,

A= 'AEG'/R,
.
.
B A,
C ANTON,

LC ANTON,

LR H3,
CH ANTON,

LC ANTON,
ZT3 ANTON,

```

Bild 3.3 Beispiele für Setzen und Löschen von Speicherzellen

3.3. Typenkennung

In einigen Fällen kann es wünschenswert sein, die Typenkennung eines Ganzwortes zu ändern. So liefert z.B. der Befehl BA im Register A stets die Typenkennung 1. Handelt es sich dabei jedoch um Alphatext, so kann es erforderlich sein, diesem Wort die Typenkennung 3 zu geben. Des weiteren ist es z.B. möglich, eine Liste von Alphawörtern mit Typenkennung 3 mit einem Wort abzuschließen, das die Typenkennung 2 hat. Bei den Tabellensuchbefehlen wird die Typenkennung als Abbruchkriterium verwendet. Bei den Befehlen TLI, TLD und TDM müssen alle Wörter der Tabelle die Typenkennung des Suchwortes haben. Bei den Befehlen TMAX und TMIN muß das letzte Wort einer Tabelle eine andere Typenkennung haben als die Wörter der Tabelle. Dieses Wort gehört nicht mehr zur Tabelle.

Die Typenkennung kann sowohl in einem der Rechenwerksregister als auch im Speicher auf einen neuen Wert gesetzt werden. Hierzu dienen die Befehle

ZTR	s	Setze Typenkennung im Register	$\langle s_2 \rangle_t := s_1$
ZT0	n	Setze Typenkennung 0	$\langle n \rangle_t := 0$
ZT1	n	Setze Typenkennung 1	$\langle n \rangle_t := 1$
ZT2	n	Setze Typenkennung 2	$\langle n \rangle_t := 2$
ZT3	n	Setze Typenkennung 3	$\langle n \rangle_t := 3$

$s_1: 0,1,2,3$
 $s_2: A,0,H,D$

Es muß jedoch beachtet werden, daß der größte Teil der Befehle in seiner Wirkung von der Typenkennung abhängig ist und ein Alphatext, der z.B. die Typenkennung 1 hat, wie eine Festkommazahl behandelt wird.

In Abschnitt 6 ist gezeigt, wie mit Hilfe der Befehle ST und STN die Typenkennung als Sprungkriterium verwendet werden kann.

4. BRINGEN

Nachstehend sind die Befehle aufgeführt, die Nicht-zahlwörter in die Rechenwerksregister bringen, gegliedert nach der Wortlänge, die gebracht wird. Soweit nicht anders vermerkt, gilt das Nachstehende für Wörter mit der Typenkennung 2 und 3.

4.1. Bringen von Ganzwörtern

Die nachstehend aufgeführten Bringebefehle holen ein Ganzwort und bringen es unverändert in das angegebene Register.

B	n	<u>B</u> ringe (nach A)	$\langle A \rangle := \langle n \rangle$
BQ	n	<u>B</u> ringe nach <u>Q</u>	$\langle Q \rangle := \langle n \rangle$
BD	n	<u>B</u> ringe nach <u>D</u>	$\langle D \rangle := \langle n \rangle$
BH	n	<u>B</u> ringe nach <u>H</u>	$\langle H \rangle := \langle n \rangle$

Der Befehl BB hat die gleiche Wirkung wie der Befehl B. Beim Befehl

BR	n	<u>B</u> ringe und <u>r</u> eserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := \langle n \rangle$
----	---	---------------------------------------	--

wird gegenüber dem Befehl B zusätzlich noch der Inhalt des Registers A im Register H sichergestellt. Der Befehl

BQB	n	<u>B</u> ringe nach <u>Q</u> und <u>b</u> ringe (nach A)	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n \rangle$
-----	---	--	--

bringt den Inhalt der Speicherzelle sowohl in das Register A als auch in das Register Q.

Eine besondere Bedeutung kommt dem Befehl BU zu. Er bringt alle Wörter unabhängig von der Typenkennung stets so, als ob sie die Typenkennung 3 (bzw. 2) haben.

BU	n	<u>B</u> ringe <u>u</u> nverändert	$\langle A \rangle := \langle n \rangle$
----	---	------------------------------------	--

unverändert bei jeder IK

Damit können auch Wörter gebracht werden, die Alphatext enthalten, jedoch die Typenkennung 0 oder 1 haben. Mit dem Befehl

BZ2	m	<u>B</u> ringe <u>z</u> wei <u>H</u> albwörter	$\langle A \rangle := \langle m, m+1 \rangle$
-----	---	--	---

können zwei Halbwörter ins Register A gebracht werden. Dabei kommt das erste (adressierte) Halbwort in die linke Hälfte des Registers und das zweite Halbwort in die rechte Hälfte. Wird eine gradzahlige Adresse angegeben, so ist die Wirkung die gleiche wie bei dem Befehl BU.

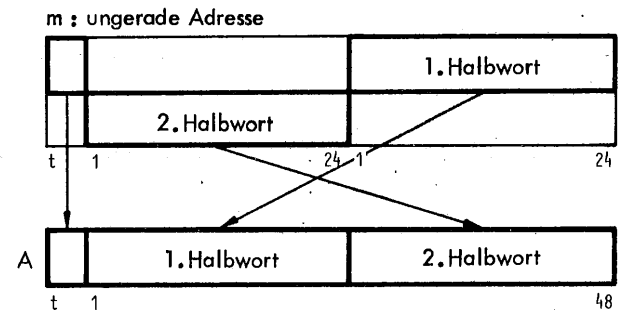
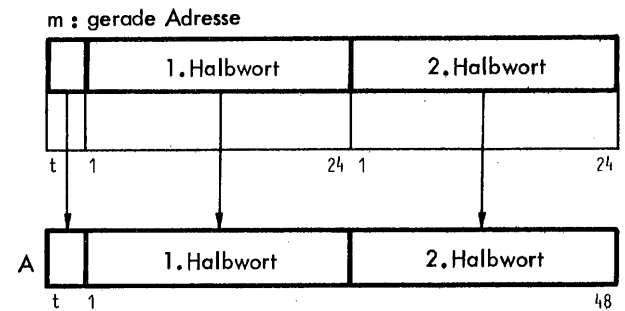


Bild 4.1 Wirkung des Befehls BZ2

Der Befehl

BC	n	<u>B</u> ringe und <u>s</u> peichere	$\langle A \rangle := \langle n \rangle$
----	---	--------------------------------------	--

bringt einen Operanden in das Register A und speichert gleichzeitig den Inhalt des Registers ab. Er tauscht also den Inhalt des Registers mit dem Inhalt der Speicherzelle aus.

Der Befehl

BL	n	<u>B</u> ringe und <u>l</u> ösche	$\langle A \rangle := \langle n \rangle$ $\langle n \rangle := +0$
----	---	-----------------------------------	---

bringt einen Operanden in das Register A und löscht gleichzeitig die Speicherzelle auf Null. Es ist zu beachten, daß die Speicherzelle die Typenkennung 0 erhält.

Bei Nichtzahlwörtern gibt es keine Vorzeichen. Bei den Befehlen

BN	n	Bringe negativ	$\langle A \rangle := -\langle n \rangle$
BNR	n	Bringe negativ und reserviere	$\langle H \rangle := \langle A \rangle$ $\langle A \rangle := -\langle n \rangle$

wird der Operand aus dem Speicher geholt und in allen 48 Binärstellen invertiert. Die weitere Wirkung ist wie bei den Befehlen B und BR.

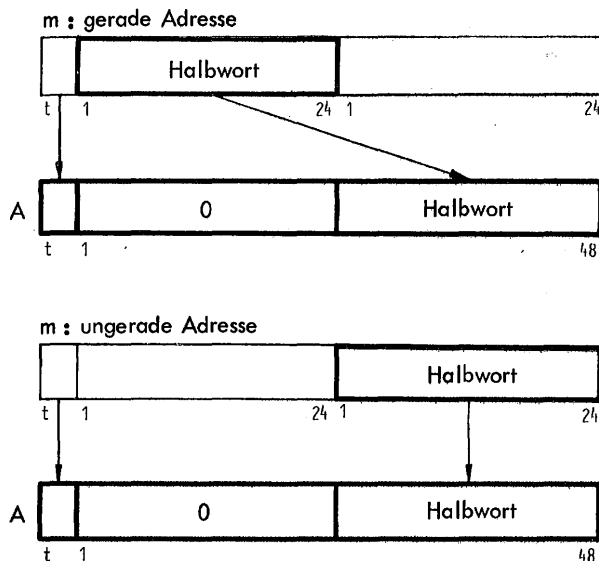


Bild 4.2 Wirkung des Befehls B2

4.2. Bringen von zwei Wörtern

Mit dem Befehl

BZ	n	Bringe zwei Wörter	$\langle A \rangle := \langle n \rangle$ $\langle Q \rangle := \langle n+2 \rangle$
----	---	--------------------	--

lassen sich zwei aufeinanderfolgende Wörter in das Rechenwerk holen, und zwar in die Register A und Q. Bei dem Befehl

BZN	n	Bringe zwei Wörter negativ	$\langle A \rangle := -\langle n \rangle$ $\langle Q \rangle := -\langle n+2 \rangle$
-----	---	----------------------------	--

werden beide Wörter in allen 48 Binärstellen invertiert. Die weitere Wirkung ist wie bei dem Befehl BZ.

4.3. Bringen von Halbwörtern

Um ein Halbwort in das Register A zu bringen, steht der Befehl

B2	m	Bringe Halbwort	$\langle A \rangle := \langle m \rangle$
----	---	-----------------	--

zur Verfügung. Er bringt das adressierte Halbwort in die rechte Hälfte des Registers A. Der Rest des Registers wird auf Null gesetzt. Der Befehl ist unabhängig von der Typenkennung der Speicherzelle. Das Register A erhält die gleiche Typenkennung wie das Wort, in dem das Halbwort steht. Die Befehle B2V und B2VN sind für Alphatexte nicht geeignet, da diese kein Vorzeichen haben.

4.4. Bringen von Drittelwörtern

Ein Drittelwort hat eine Länge von 16 Bits. Mit dem Befehl

B3	m	Bringe Drittelwort	$\langle A \rangle := \langle m \rangle_{9-24}$
----	---	--------------------	---

wird ein Halbwort adressiert. Aus diesem Halbwort werden die rechten 16 Bits in das Register A gebracht, und zwar in die rechten 16 Bits. Der Rest wird mit 0 aufgefüllt.

Der Befehl ist unabhängig von der Typenkennung der Speicherzelle.

Es ist darauf zu achten, daß das Register A die Typenkennung 1 erhält.

Der Befehl B3V ist für Alphatexte nicht geeignet, da diese kein Vorzeichen haben.

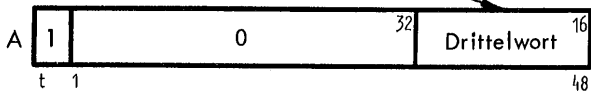
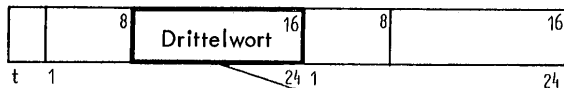
4.5. Zeichenweise bringen

Mit Hilfe des Befehls

BNZ	i_L	i_R	Bringe nächstes Zeichen	$\langle A \rangle := \text{gem. } i_R \text{ und } i_L$
-----	-------	-------	-------------------------	--

- a: $\langle i_R \rangle + \text{mod}2$
lauf. Adr. eines Wortes der Liste
- b: $\langle i_L \rangle_{17-24}$
lauf. Nummer eines Zeichens im Wort (0,1,...)

m : gerade Adresse



m : ungerade Adresse

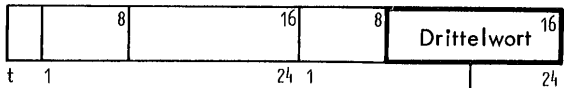


Bild 4.3 Wirkung des Befehls B3

ist es möglich, jeweils ein Zeichen aus einem Wort in das Register A zu holen. Das Zeichen kann eine Länge von 4, 6, 8 oder 12 Bits haben. Das Zeichen steht rechts im Register A. Der Rest wird auf 0 gesetzt.

Vor der Ausführung des Befehls muß die Adresse des Ganzwortes, in dem das Zeichen steht, in der Indexzelle i_R stehen. In der Indexzelle i_L muß stehen, wie lang das Zeichen ist und welches Zeichen es innerhalb des Ganzwortes ist, wobei das linke Zeichen die laufende Nummer 0 hat.

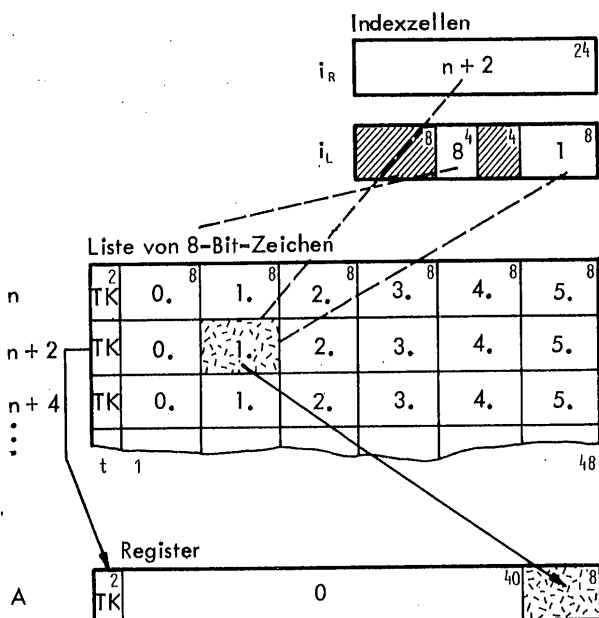


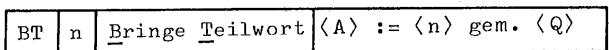
Bild 4.4 Beispiel für Befehl BNZ

Gleichzeitig wird durch diesen Befehl die laufende Nummer des Zeichens um 1 erhöht. Wird dabei das letzte (rechte) Zeichen eines Wortes erreicht, so wird auf das linke Zeichen des nächsten Wortes weitergeschaltet, indem die laufende Nummer des Zeichens auf 0 gesetzt und die Adresse in der Indexzelle i_R um 2 erhöht wird.

Dieser Befehl erhält seine besondere Bedeutung, wenn Alphatexte zeichenweise verarbeitet werden. Er ist unabhängig von der Typenkennung. Näheres dazu ist im Abschnitt 9 beschrieben.

4.6. Teilwort bringen

Um einen beliebigen Ausschnitt aus einem Ganzwort in das Register A zu bringen, steht der Befehl



zur Verfügung. Welcher Teil des Ganzwortes gebracht wird, wird durch eine Maske bestimmt, die vor der Ausführung des Befehls BT im Register Q stehen muß. Alle Binärstellen der adressierten Speicherzelle, bei denen im Register Q eine "0" steht (Null-Feld der Maske) werden in das Register A gebracht. Alle Binärstellen des Registers A, bei denen im Register Q ein "L" steht (L-Feld der Maske), werden auf "0" gelöscht.

Anschließend wird der Inhalt des Registers A um soviel Stellen nach rechts geschiftet, wie im Register Q rechts aufeinanderfolgende "L"-Stellen sind. Links werden dabei Nullen nachgezogen.

Dieser Befehl ist besonders geeignet, um bestimmte Teile eines Wortes zu verarbeiten. Er arbeitet unabhängig von der Typenkennung. Näheres dazu ist im Abschnitt 10 beschrieben.

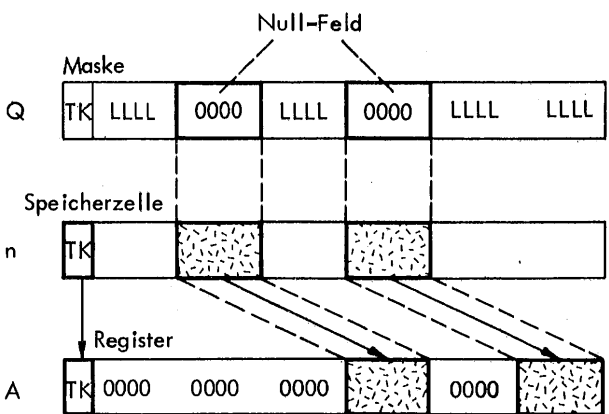


Bild 4.5 Beispiel zum Befehl BT

5. SPEICHERN

Nachstehend sind die Befehle aufgeführt, die Nichtzahlwörter aus den Rechenwerksregistern in den Speicher bringen, gegliedert nach der Wortlänge, die abgespeichert wird. Soweit nicht anders vermerkt, gilt das Nachstehende für Wörter mit der Typenkennung 2 und 3.

speichert den Inhalt des Registers A ab und bringt gleichzeitig den Inhalt der Speicherzelle in das Register A. Er tauscht also die Inhalte des Registers A mit der Speicherzelle n aus.

Bei Nichtzahlwörtern gibt es kein Vorzeichen. Bei dem Befehl

CN	n	Speichere negativ	$\langle n \rangle := -\langle A \rangle$
----	---	-------------------	---

wird der Inhalt des Registers A erst in allen Binärstellen invertiert. Die weitere Wirkung ist wie bei den Befehlen C und CR.

Da Nichtzahlwörter keine Markenstelle haben, sind die Befehle CMT, CMR und CMC nur sehr bedingt von Bedeutung.

5.1. Speichern von Ganzwörtern

Die nachstehend aufgeführten Befehle nehmen ein Ganzwort aus einem der angegebenen Register und bringen es unverändert in den Kernspeicher.

C	n	Speichere	$\langle n \rangle := \langle A \rangle$
CQ	n	Speichere aus Q	$\langle n \rangle := \langle Q \rangle$
CD	n	Speichere aus D	$\langle n \rangle := \langle D \rangle$
CH	n	Speichere aus H	$\langle n \rangle := \langle H \rangle$

Sie sind das Gegenstück zu den im Abschnitt 3.1. aufgeführten Bringebefehlen. Der Befehl CB hat die gleiche Wirkung wie der Befehl C.

Das Gegenstück zum Befehl BR ist der Befehl

CR	n	Speichere und bringe Reserve	$\langle n \rangle := \langle A \rangle$ $\langle A \rangle := \langle H \rangle$
----	---	------------------------------	--

Er speichert den Inhalt des Registers A ab und bringt den (z.B. mit BR reservierten) Wert aus dem Register H in das Register A.

Eine besondere Bedeutung kommt dem Befehl CU zu. Er ist das Gegenstück zum Befehl BU und speichert ebenfalls alle Wörter unabhängig von der Typenkennung stets so, als ob sie die Typenkennung 3 (bzw. 2) haben.

CU	n	Speichere unverändert	$\langle n \rangle := \langle A \rangle$
----	---	-----------------------	--

unverändert bei jeder TK

Damit können auch Wörter gespeichert werden, die Alphatext enthalten, jedoch die Typenkennung 0 oder 1 haben. Der Befehl

BC	n	Bringe und speichere	$\langle A \rangle := \langle n \rangle$
----	---	----------------------	--

5.2. Speichern von zwei Wörtern

Die Inhalte der Register A und Q können gleichzeitig in zwei aufeinanderfolgende Ganzwörter abgespeichert werden. Hierzu dient der Befehl

CZ	n	Speichere zwei Wörter	$\langle n \rangle := \langle A \rangle$ $\langle n+2 \rangle := \langle Q \rangle$
----	---	-----------------------	--

Er ist das Gegenstück zum Befehl BZ.

5.3. Speichern von Halbwörtern

Die rechte Hälfte des Registers A kann in einem Halbwort im Kernspeicher abgelegt werden. Hierzu dient der Befehl

C2	m	Speichere Halbwort	$\langle m \rangle := \langle A \rangle$ 25-48
----	---	--------------------	--

Das Halbwort wird unverändert eingesetzt

Er speichert also ein Halbwort. Die andere Hälfte des Ganzwortes im Speicher und die Typenkennung bleiben unverändert. Der Befehl ist unabhängig von der Typenkennung.

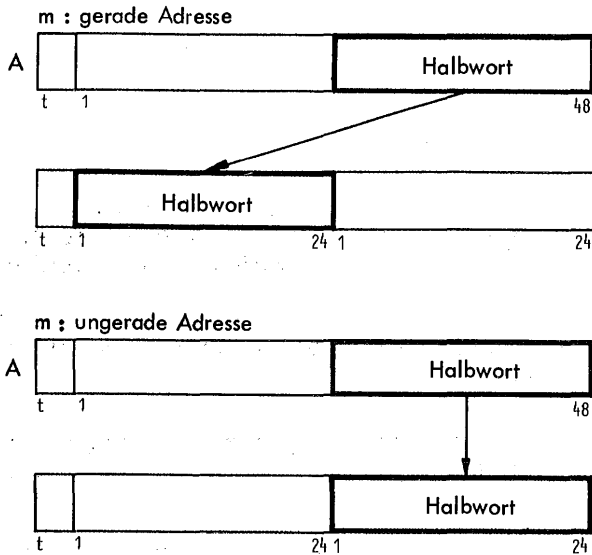


Bild 5.1 Wirkung des Befehls C2

5.4. Speichern von Drittelwörtern

Ein Drittelwort hat eine Länge von 16 Bits. Mit dem Befehl

C3	m	Speichere Drittelwort	$\langle m \rangle_{8-24} := \langle A \rangle_{33-48}$
----	---	-----------------------	---

Das Drittelwort wird unverändert eingesetzt

werden die rechten 16 Bits des Registers A in die rechten 16 Bits des adressierten Halbwortes gebracht. Der andere Teil des Ganzwortes im Speicher und die Typenkennung bleiben unverändert. Der Befehl ist unabhängig von der Typenkennung.

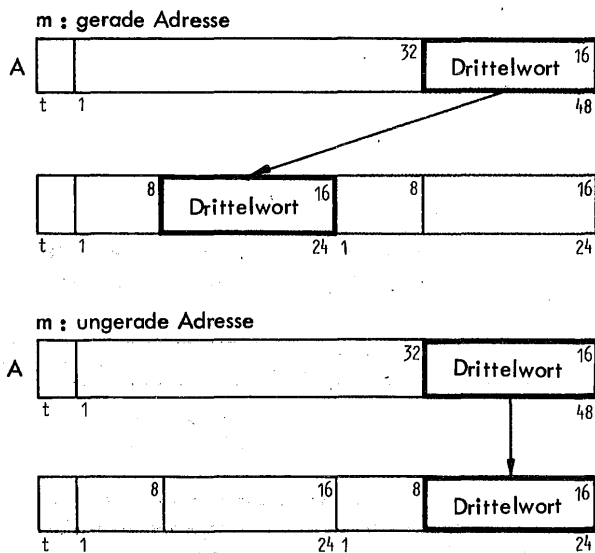


Bild 5.2 Wirkung des Befehls C3

5.5. Zeichenweise speichern

Das Gegenstück zum Befehl BNZ ist der Befehl

CNZ	i_L	i_R	Speichere nächstes Zeichen	Zeichen gem. i_R u. i_L abspeich.
-----	-------	-------	----------------------------	---------------------------------------

Mit seiner Hilfe ist es möglich, ein Zeichen, das rechtsbündig im Register A steht, im Kernspeicher abzuspeichern. Das Zeichen kann eine Länge von 4, 6, 8 oder 12 Bits haben. Der andere Teil der Speicherzelle bleibt unverändert.

Vor der Ausführung des Befehls muß die Adresse des Ganzwortes, in dem das Zeichen abgelegt werden soll, in der Indexzelle i_R stehen. In der Indexzelle i_L muß stehen, wie lang das Zeichen ist und welches Zeichen es innerhalb des Ganzwortes sein soll, wobei das linke Zeichen die laufende Nummer 0 hat.

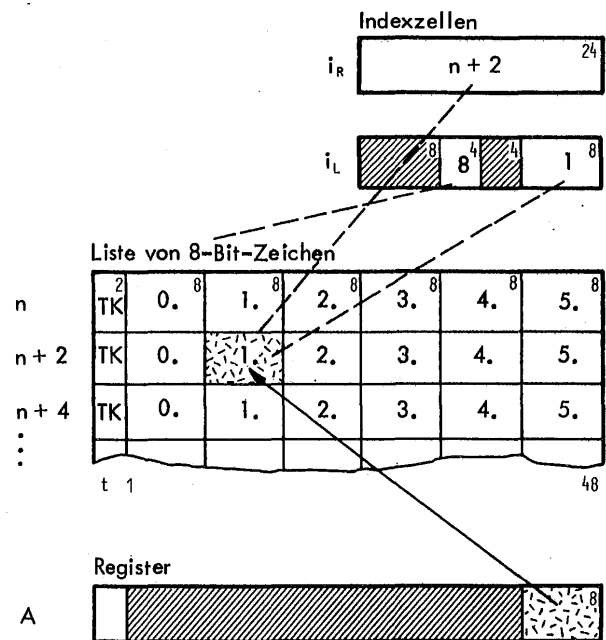


Bild 5.3 Wirkung des Befehls CNZ

Gleichzeitig wird durch den Befehl die laufende Nummer des Zeichens um 1 erhöht. Wird dabei das letzte (rechte) Zeichen eines Wortes erreicht, so wird auf das linke Zeichen des nächsten Ganzwortes weitergeschaltet, indem die laufende Nummer des Zeichens auf 0 gesetzt wird und die Adresse in der Indexzelle i_R um 2 erhöht wird.

Dieser Befehl erhält seine besondere Bedeutung, wenn Alphanetze zeichenweise verarbeitet werden. Er ist unabhängig von der Typenkennung. Näheres dazu ist im Abschnitt 9 beschrieben.

5.6. Teilwort speichern

Das Gegenstück zum Befehl BT ist der Befehl

CT	n	Speichere Teilwort	$\langle n \rangle := \langle A \rangle \text{ gem. } \langle Q \rangle$
----	---	--------------------	--

Er speichert einen bestimmten Teil des Registers A im Kernspeicher ab. Welcher Teil des Registers A abgespeichert wird und an welcher Stelle der Speicherzelle dies erfolgt, wird durch eine Maske bestimmt, die vor der Ausführung des Befehls CT im Register Q stehen muß.

Bei dem Befehl BT wurde nach dem Bringen des Teilwortes in das Register A das Register A nach rechts geschiftet. Bei dem Befehl CT wird zuerst ein Schift nach links im Kreis gemacht, und zwar um soviel Stellen, wie im Register Q (Maske) rechts aufeinanderfolgende "L"-Stellen sind. Rechts werden die links herausgeschifteten Stellen wieder hineingeschiftet (Kreisschift).

Danach werden die Binärstellen des Registers A, bei denen im Register Q eine "0" steht (Null-Feld der Maske), in die Speicherzelle gebracht. Die anderen Stellen der Speicherzelle bleiben unverändert.

Schließlich wird der Inhalt des Registers A wieder in seine ursprüngliche Lage zurückgeschiftet.

Als Nebenwirkung tritt hierbei auf, daß der neue Inhalt der Speicherzelle gleichzeitig im Register D steht.

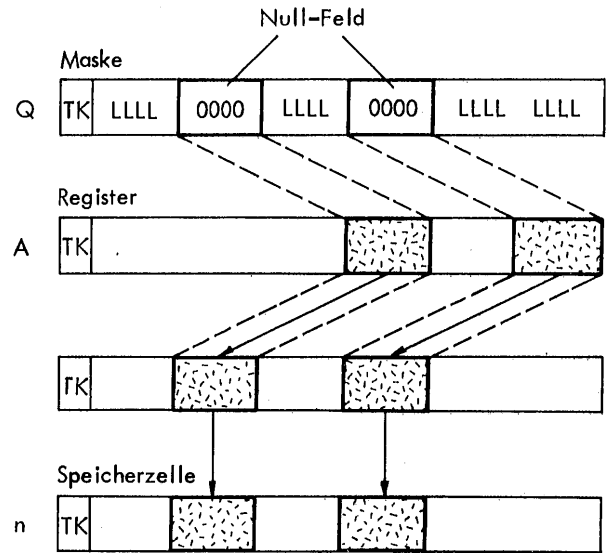


Bild 5.4 Beispiel zum Befehl CT

Dieser Befehl ist besonders geeignet, um bestimmte Teile eines Wortes zu verarbeiten. Er arbeitet unabhängig von der Typenkennung. Näheres dazu ist im Abschnitt 10 beschrieben.

L

6. ABFRAGEN - VERZWEIGUNGEN - SPRÜNGE

Zur Verzweigung innerhalb der Programme und zur Schleifenbildung stehen die Sprungbefehle zur Verfügung. Sie ermöglichen es,

- den Inhalt des Registers A mit Null zu vergleichen
- den Inhalt des Registers A mit dem Inhalt des Registers H zu vergleichen
- die Typenkennung in einem Rechenwerksregister abzufragen
- ein Bit des Registers A abzufragen

und auf Grund des Ergebnisses auf eine andere Stelle des Programms zu springen und dort fortzufahren.

Die nachstehend nicht erwähnten Befehle SGG0 und SK0 sind bei Nichtzahlwörtern nicht sinnvoll. Bezüglich der Alarm-Sprungbefehle SAA und SAT sowie der Merklichter-Sprungbefehle wird auf den Abschnitt "Sonstige Operationen" verwiesen.

6.2. Vergleich Register A und H

Um den Inhalt des Registers A mit dem Inhalt des Registers H zu vergleichen, stehen die Befehle

SI	m	Sprunge wenn <u>identisch</u>	$\langle A \rangle = \langle H \rangle$
SN	m	Sprunge wenn <u>nicht identisch</u>	$\langle A \rangle \neq \langle H \rangle$
SGG	m	Sprunge wenn <u>größer gleich</u>	$\langle A \rangle \geq \langle H \rangle$
SG	m	Sprunge wenn <u>größer</u>	$\langle A \rangle > \langle H \rangle$
SKG	m	Sprunge wenn <u>kleiner gleich</u>	$\langle A \rangle \leq \langle H \rangle$
SK	m	Sprunge wenn <u>kleiner</u>	$\langle A \rangle < \langle H \rangle$

zur Verfügung. Nichtzahlwörter werden dabei verglichen, als ob sie ganze, vorzeichenlose Binärzahlen wären. Identisch (oder gleich) sind sie nur dann, wenn die beiden Register in allen Binärstellen übereinstimmen.

Bei erfüllter Sprungbedingung wird auf die im Adressenteil des Befehls angegebene Adresse gesprungen und das Programm an dieser Stelle fortgesetzt.

Hat eine der Vergleichsgrößen nicht die Typenkennung 2 oder 3, so wird der Vergleich ebenfalls wie bei Nichtzahlwörtern durchgeführt. Die Höhen der beiden Typenkennungen bestimmt die Art des Vergleichs.

Im Bild 6.1 ist mit Hilfe der Konstantenschreibweise der TAS-Sprache an einigen Beispielen gezeigt, wann die Bedingung "größer" erfüllt ist (Befehl SG).

6.1. Vergleich Register A mit Null

Der Inhalt des Registers A kann mit Null verglichen werden. Dazu stehen die Befehle

SIO	m	Sprunge wenn <u>identisch 0</u>	$\langle A \rangle = 0$
SNO	m	Sprunge wenn <u>nicht 0</u>	$\langle A \rangle \neq 0$
SGO	m	Sprunge wenn <u>größer 0</u>	$\langle A \rangle > 0$
SKGO	m	Sprunge wenn <u>kleiner gleich 0</u>	$\langle A \rangle \leq 0$

zur Verfügung. Dabei gilt für Nichtzahlwörter, daß sie nur dann Null sind, wenn alle Binärstellen 0 sind. In allen anderen Fällen sind sie größer als Null.

Aus dem Vorstehenden ergibt sich, daß der Befehl SGO bzw. SKGO gleichwertig ist mit dem Befehl SIO bzw. SNO.

Ist die Bedingung erfüllt, so wird auf die im Adressenteil angegebene Adresse gesprungen und das Programm an dieser Stelle fortgesetzt.

Register A	größer	Register H
'5'	>	'4'
'5'	>	'09'
'5'	>	'9'/R
'A3'	>	'81'
'2'/L	>	'3'
100	>	20/2
-100	>	100/3

Bild 6.1 Beispiele für die Bedingung "größer als"



6.3. Abfrage eines Bits

Für die Abfrage einzelner Binärstellen gibt es zwei Möglichkeiten. Mit den Befehlen

SR	m	Springe wenn <u>rechtes</u> Bit in A <u>g</u> esetzt	$\langle A \rangle_{48} = L$
SRN	m	Springe wenn <u>rechtes</u> Bit in A <u>n</u> icht gesetzt	$\langle A \rangle_{48} = 0$

wird das rechte Bit im Register abgefragt. Bei erfüllter Bedingung wird auf die im Adressenteil des Befehls angegebene Adresse gesprungen. Mit dem Befehl

SBIT	p s	Springe wenn <u>Bit</u> <u>g</u> esetzt	$\langle s_1 \rangle_{s_2} = L$
------	-----	---	---------------------------------

p: +0...+127
 s_1 : A, Q, D oder H
 s_2 : 1...48

kann in den 4 Rechenwerksregistern jedes der 48 Bits abgefragt werden, ob es den Wert "L" hat. Ist dies der Fall, so wird um p Befehle vor- oder zurückgesprungen.

Die vorstehenden Befehle sind unabhängig von der Typenkennung.

6.4. Abfrage der Typenkennung

Die Typenkennung eines Wortes kann ein Kriterium für eine Programmverzweigung oder den Abbruch einer Schleife sein.

So kann z. B. die Typenkennung zur Kennzeichnung verschiedener Alphanetze verwendet oder eine Liste von Alphanetzen durch ein Wort mit anderer oder bestimmter Typenkennung abgeschlossen werden.

Bei den Tabellenbefehlen TMAX und TMIN wird die vorstehend beschriebene Art angewendet.

Für die Abfrage der Typenkennung in den Rechenwerksregistern stehen die Befehle

ST	p s	Springe wenn <u>Typenkennung</u> $\langle s_2 \rangle = s_1$
STN	p s	Springe wenn <u>Typenkennung</u> $\langle s_2 \rangle \neq s_1$ <u>n</u> icht

p: +0...+127
 s_1 : 0, 1, 2 oder 3 (IK)
 s_2 : A, Q, D oder H

zur Verfügung. Bei erfüllter Sprungbedingung wird um p Befehle vor- bzw. zurückgesprungen.

7. SCHIFTEN

Das Schiften ist im Abschnitt "Sonstige Operationen", Abschnitt 1, ausführlich beschrieben.

Im Rechenwerk kann in den Registern A und Q geschifft werden. Für das Schiften steht der Befehl

SH	s p	<u>S</u> chifte	$\langle s_1 \rangle := \langle s_1 \rangle$ geschifft
----	-----	-----------------	--

p: Anzahl der Schifftschritte
+0...+127

s₁: leer = kein Schift
A = Register A
Q = Register Q
AQ = Register A und Q getrennt } Kurzschrift
Z = Register A und Q zusammen } Langschrift

s₂: leer = Rechtsschift
L = Linksschift } Schiftrichtung

s₃: leer = gestreckter Schift
K = Kreisschift

s₄: leer = ohne Rundung
R = mit Rundung } Rundung nach dem Schiften

s₅: leer = abhängig von der Typenkennung
U = unabhängig von der Typenkennung

s₆: leer = nicht zählen
B = zählen der aus A geschifteten L-Bits

zur Verfügung. Mit dem Parameter p wird angegeben, um wieviel Binärstellen geschifft wird. Die Spezifikation s gibt die Art des Schiftes an. Es gibt folgende Möglichkeiten, die entsprechend kombiniert werden können.

7.1. Register

Es kann im Register A, im Register Q und in beiden gleichzeitig, aber getrennt geschifft werden. Geschifft werden alle 48 Binärstellen. Eine weitere Möglichkeit besteht darin, das Register A und Q als gemeinsames doppelt langes Register zu betrachten. Es ist dann der Buchstabe Z (zusammen) anzugeben. In diesem Fall werden 96 Binärstellen geschifft.

Wird die Spezifikation Z angegeben und ist in einem der Register ein Zahlwort (Typenkennung 0 oder 1), so wird der Vergleich wie bei Nichtzahlwörtern durchgeführt. Die höhere der beiden Typenkennungen bestimmt, wie der Schift ausgeführt wird.

7.2. Schiftrichtung

Das Schiften kann innerhalb der Register nach rechts und nach links erfolgen. Wird nichts angegeben, so wird nach rechts geschifft. Soll nach links geschifft werden, so ist der Buchstabe L anzugeben.

Beim Schiften gehen die herausfallenden Binärstellen verloren. Auf der anderen Seite des Registers werden Nullen nachgezogen.

7.3. Kreisschift

Ohne Angabe wird der Schift gestreckt ausgeführt, d.h. auf der einen Seite des Registers (der Register) fallen Binärstellen heraus und gehen verloren, auf der anderen Seite werden Nullen nachgezogen.

Es ist jedoch möglich, durch Angabe des Buchstabens K (Kreisschift) die auf der einen Seite herausgeschifteten Stellen auf der anderen Seite wieder hineinzuschiften. Die beiden Enden des Registers werden also miteinander zu einem Kreis verbunden. Es gehen dabei keine Binärstellen verloren.

7.4. Typenkennung

Der Schifftbefehl ist abhängig von der Typenkennung. Bei der Typenkennung 0 und 1 (Zahlwörter) wird nach Festkommaart geschifft, bei Typenkennung 2 und 3 (Nichtzahlwörter) wird ein Bitmuster geschifft. Haben bei der Spezifikation Z die beiden Register verschiedene Typenkennungen, so bestimmt die höhere Typenkennung wie geschifft wird.

Wird bei der Spezifikation der Buchstabe U angegeben, so wird unabhängig von der Typenkennung stets ein Bitmuster wie bei Nichtzahlwörtern geschifft. Damit wird es möglich, Alphatexte zu schiften, die mit der Typenkennung 0 oder 1 versehen sind.

7.5. Zählen der besetzten Bits

Die Angabe des Buchstabens B bei der Spezifikation hat zur Folge, daß die aus dem Register A rechts oder links herausgeschifteten Bits, die auf "L" gesetzt sind, im Register Y (Schifftzähler) gezählt werden. Mit Hilfe der Befehle R und RLR ist es möglich, auf den Inhalt des Registers Y zurückzugreifen.

Der Buchstabe B kann beliebig kombiniert werden. Da jedoch nur die Bits des Registers A gezählt werden, ist B bei einer Spezifikation Q nur insofern von Bedeutung, als der Inhalt des Registers Y auf Null gesetzt wird.

7.6. Beispiele

In den Bildern 7.1 und 7.2 ist in einem Beispiel gezeigt, wie mit Hilfe von Schifftbefehlen bei einer Zahl die führenden Nullen durch Leertasten (□) ersetzt werden, damit sie bei der Ausgabe nicht erscheinen. Die Zahl ist im Zentralcode, also in Oktaden, dargestellt.

Um die Zahl wieder in ihre ursprüngliche Lage zurückzuschiften, wurde in der Indexzelle X1 gezählt, wieviel Schifftschritte gemacht sind. Beim Linksschift wird hochgezählt, beim Rechtsschift heruntergezählt. Zum Schluß wird der letzte Schifftbefehl mit der Zählgröße modifiziert. Damit wird die Zahl der Schifftschritte, um die die Zahl nach links geschifft ist, auf den Befehl SH Z 0 addiert, und die Zahl wird damit in ihre ursprüngliche Position zurückgeschifft. An Stelle der führenden Nullen sind Leertasten eingesetzt.

Im Beispiel Bild 7.3 ist gezeigt, wie in eine Zahl, die im Zentralcode in zwei Wörtern dargestellt ist, hinter die Tausenderstelle ein Punkt und das Dezimalkomma eingesetzt werden. Führende Nullen sind bereits durch Leertasten ersetzt.

Die Addition bei Nichtzahlwörtern ist im Abschnitt 8 beschrieben. Hier sei kurz der im Beispiel verwendete Befehl SBA erläutert. Im Zentralcode haben die Zeichen folgende Werte:

- = 175
- . = 169
- , = 170

Um von der Leertaste auf den Punkt zu kommen, muß also von der Oktade der Wert 6 subtrahiert werden und um auf das Komma zu kommen der Wert 5. Im Beispiel wurde

$$\text{SBA } \text{''}\square\text{''} - \text{''}\cdot\text{''} \hat{=} \text{SBA } 6$$

$$\text{SBA } \text{''}\square\text{''} - \text{''}\cdot\text{''} \hat{=} \text{SBA } 5$$

geschrieben, d.h. es wurden die Werte 6 bzw. 5 subtrahiert. Dabei wird im Register A nur die rechte Oktade verändert, so daß die anderen Oktaden unverändert bleiben.

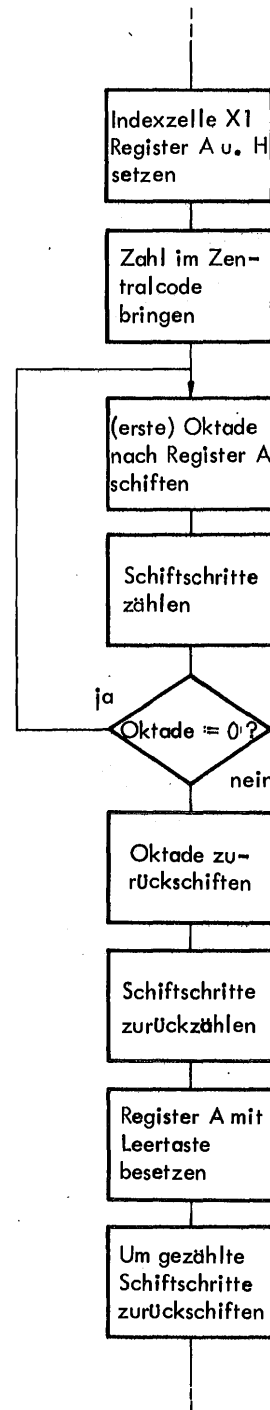
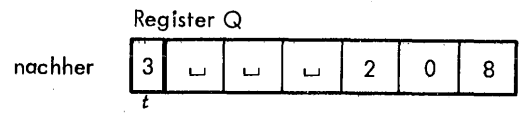
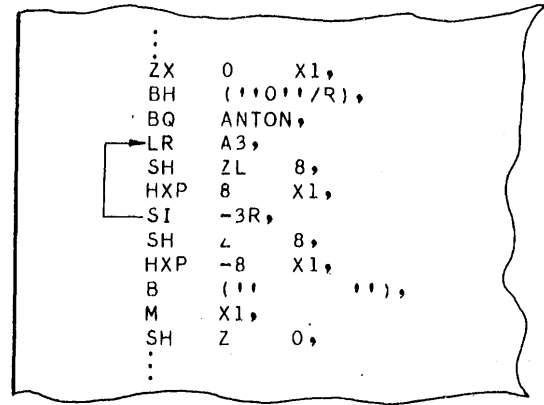
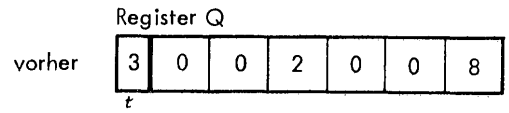


Bild 7.2 Ablaufdiagramm zum Beispiel Bild 7.1



□ = Leertaste

Bild 7.1 Beispiel Nullenunterdrückung

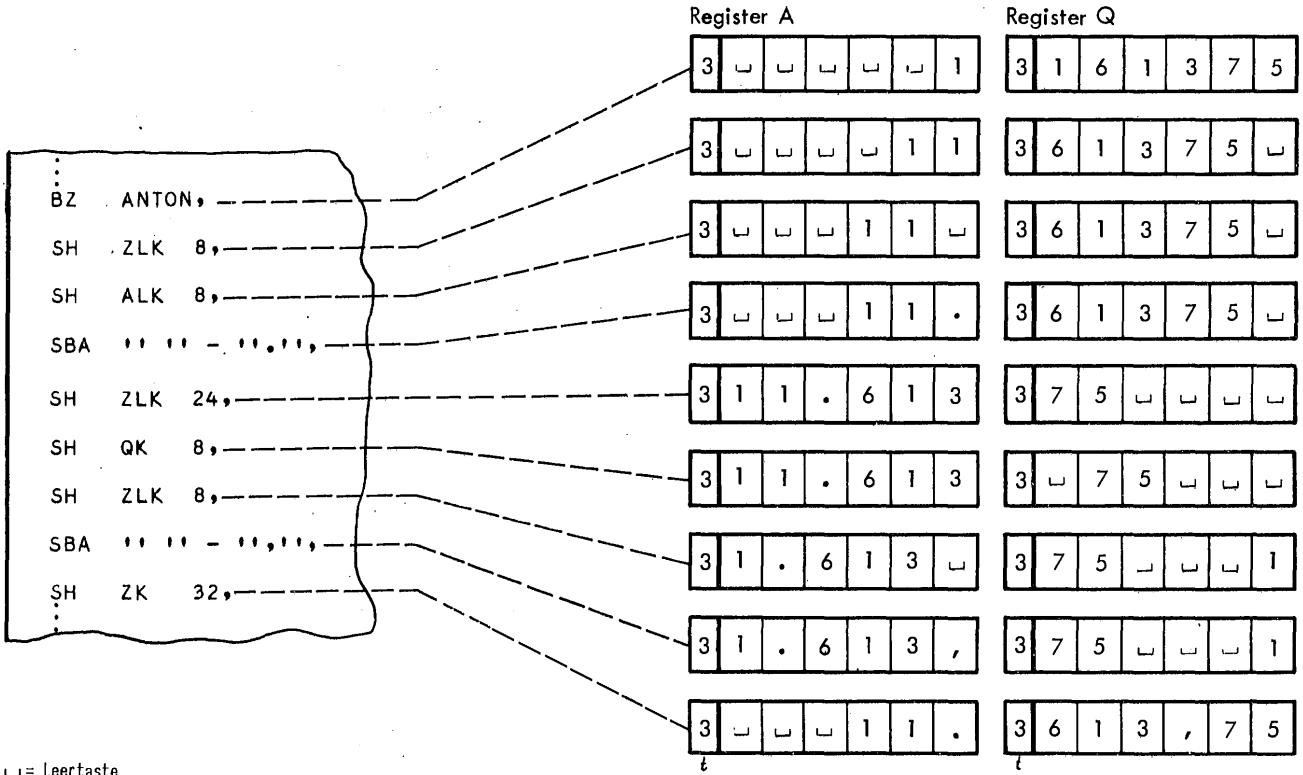


Bild 7.3 Beispiel für Einsetzen von Punkt und Komma

```

    :
    LR Q3,
    B ANTON,
    SH ZLK 32,
    BH ('00'),
    SN A,
    SH ZK 32,
    SBA '0' - '...',
    SH ZK 8,
    SBA '0' - '...',
    SH ZK 8,
    S A+1,
    A= SH ZK 48,
    SH AL 8,
    AA ' ',
    SH ZL 16,
    :
  
```

Bild 8.4 Programm zum Beispiel Bild 8.3

Zahl = 56700 Pfennig

	Register A					Register Q					
	0	5	6	7	0	0	0	0	0	0	0
SH ZLK 32	0	0	0	0	0	0	0	0	0	0	0
SH ZK 32	0	5	6	7	0	0	0	0	0	0	0
SBA	0	5	6	7	0	-	0	0	0	0	0
SH ZK 8	0	5	6	7	0	-	0	0	0	0	0
SBA	0	5	6	7	-	-	0	0	0	0	0
SH ZK 8	0	5	6	7	-	-	0	0	0	0	0
SH AL 8	0	5	6	7	0	-	-	0	0	0	0
AA	0	5	6	7	,	-	-	0	0	0	0
SH ZL 16	5	6	7	,	-	-	0	0	0	0	0

Zahl = 56783 Pfennig

	Register A						Register Q					
	0	5	6	7	8	3	0	0	0	0	0	0
SH ZLK 32	8	3	0	5	6	7	0	0	0	0	0	0
SH ZK 48	0	5	6	7	8	3	0	0	0	0	0	0
SH AL 8	0	5	6	7	8	3	0	0	0	0	0	0
AA	0	5	6	7	,	8	3	0	0	0	0	0
SH ZL 16	5	6	7	,	8	3	0	0	0	0	0	0

0 = Oktadenwert 0 ('00')

Bild 8.5 Registerstände zum Beispiel aus Bild 8.4

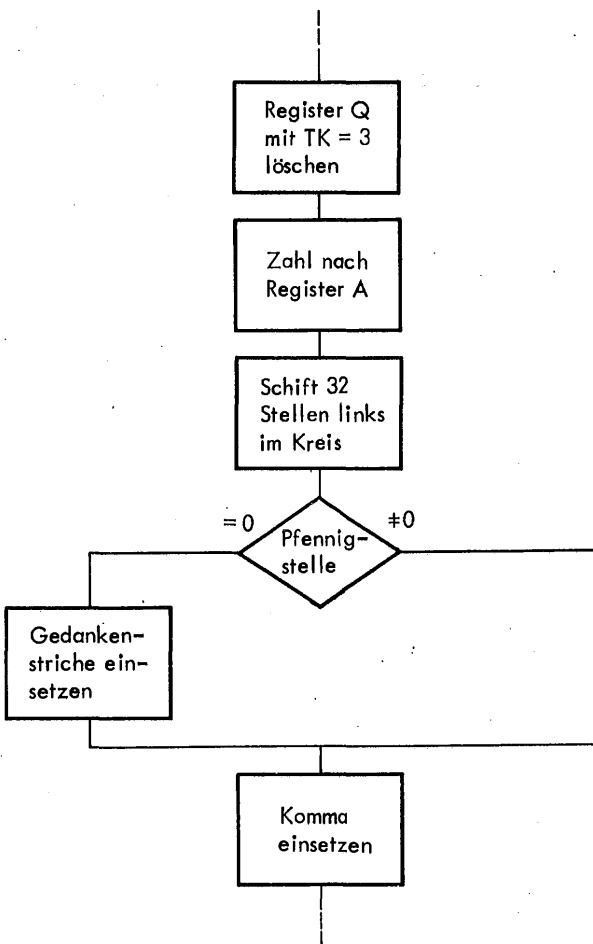


Bild 8.3 Ablaufdiagramm eines Beispiels

IAS - Handbuch

Dez. 68



9. ZEICHENVERARBEITUNG

Neben den Befehlen, die Halb- und Drittelwörter verarbeiten, gibt es Befehle, die es ermöglichen, einzelne Zeichen eines Wortes zu verarbeiten. Dabei können die Zeichen eine Länge von 12 Bits, 8 Bits und 4 Bits haben.

Außerdem stehen noch Teilwortbefehle zur Verfügung, die es erlauben, einen beliebigen Ausschnitt aus einem Ganzwort zu verarbeiten. Diese Teilwortbefehle sind, da sie eine etwas andere Funktion haben, im Abschnitt 10 beschrieben.

Die Indexzellen i_R und i_L müssen vor der Ausführung der Befehle BNZ und CNZ die gewünschten Werte enthalten. Um die Indexzellen zu setzen, kann z.B. die Befehlsfolge

TCB	(adresse/A),	i_R, i_L : Indexadressen
XC	$i_R,$	adresse: Adresse des Wortes
XBA	'f00b'/R,	f: Länge der Zeichen 4,6,8 oder C
XC	$i_L,$	b: Nummer, der Zeichen max. C,8,6 oder 4

verwendet werden. Damit wird ein Zeichen gebracht bzw. abgespeichert, das die Länge f hat und im Wort mit der Adresse "adresse" steht. Innerhalb dieses Wortes ist es das Zeichen mit der Nummer b.

9.1. Bringen und Speichern

Um ein Zeichen aus dem Speicher in das Register A zu holen bzw. ein Zeichen aus dem Register A im Speicher abzulegen, stehen die Befehle

BNZ	i_L	i_R	Bringe nächstes Zeichen	Bringen bzw. Abspeichern gem. a und b
CNZ	i_L	i_R	Speichere nächstes Zeichen	

- a: $\langle i_R \rangle + \text{mod}2$
lauf. Adr. eines Wortes der Liste
- b: $\langle i_L \rangle_{17-24}$
lauf. Nr. eines Zeichens im Wort (0, 1, ...)
- d: $(48/f) - 1$
max. Zeichennummer
- f: $\langle i_L \rangle_{9-12}$
Anzahl der Bits pro Zeich. (4, 6, 8 oder 12)
andere Bits $\langle i_L \rangle$ bedeutungslos

zur Verfügung.

Die beiden Befehle werden im Adressenteil wie Indexadressen angegeben. In der Indexzelle i_R steht die Adresse des Ganzwortes, aus dem bzw. in das das Zeichen gebracht werden soll. Die Indexzelle i_L enthält die Angabe, wie lang das Zeichen ist. Es kann 4, 6, 8 und 12 Bits lang sein. Außerdem enthält sie die Angabe, das wievielte Zeichen im adressierten Ganzwort es ist. Dabei hat das linke Zeichen die Nummer 0. Da ein Wort, je nach Zeichenlänge, 12, 8, 6 oder 4 Zeichen aufnehmen kann, ist die maximale Zeichennummer 11, 7, 5 oder 3.

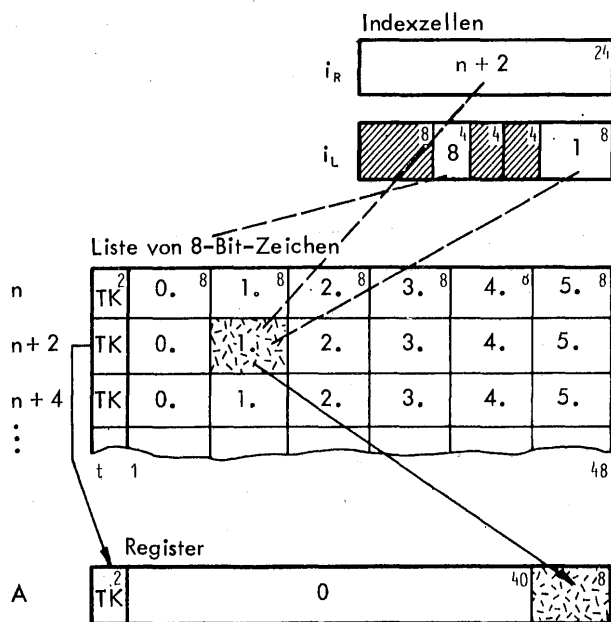
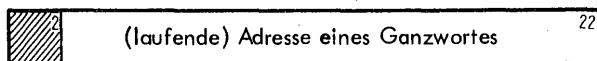


Bild 9.2 Beispiel für Befehl BNZ

Indexzelle i_R



Indexzelle i_L

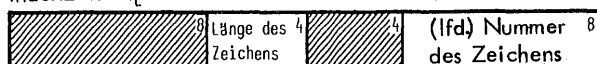


Bild 9.1 Inhalte der Indexzellen

Eine weitere, äußerst bedeutende Wirkung der Befehle BNZ und CNZ ist es, daß bei jeder Ausführung die Zeichenadresse um 1 hochgezählt wird. Das hat zur Folge, daß nach der Ausführung des Befehls die beiden Indexzellen so gesetzt sind, daß sie auf das nächste Zeichen weisen. Dies geschieht dadurch, daß die Zeichennummer um 1 erhöht wird. Ist jedoch bereits die letzte Zeichennummer des Wortes erreicht, so wird die Zeichennummer auf 0 gesetzt und die Adresse des Wortes um 2 erhöht. Diese Wirkung ist im Bild 9.3 gezeigt.

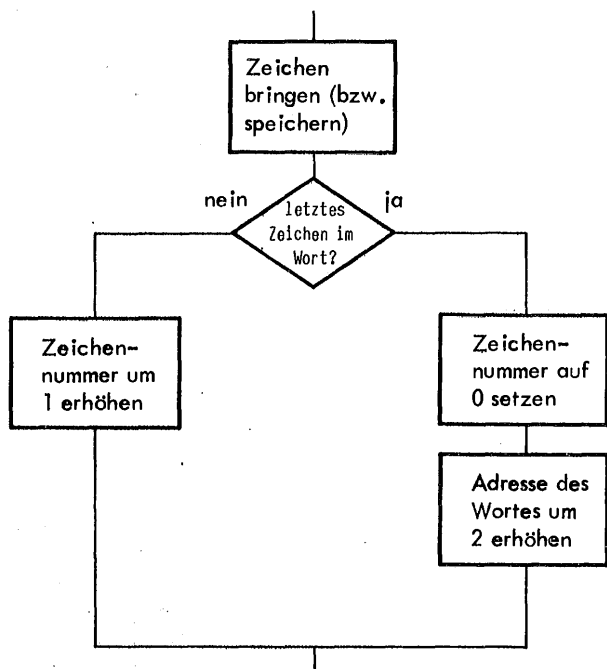
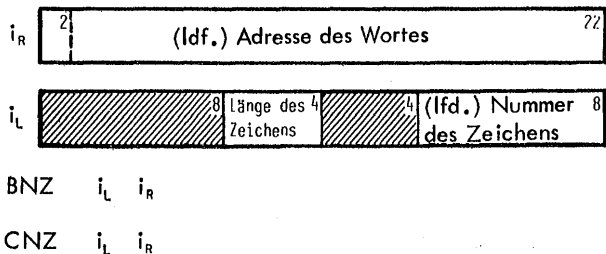


Bild 9.3 Wirkung des Befehls BNZ bzw. CNZ

Dies ermöglicht es, durch wiederholtes Anwenden des Befehls BNZ aus einer Liste von Ganzwörtern, die Zeichen gleicher Länge enthalten, ein Zeichen nach dem anderen aus dem Speicher in das Register A zu holen, um sie dort zu untersuchen.

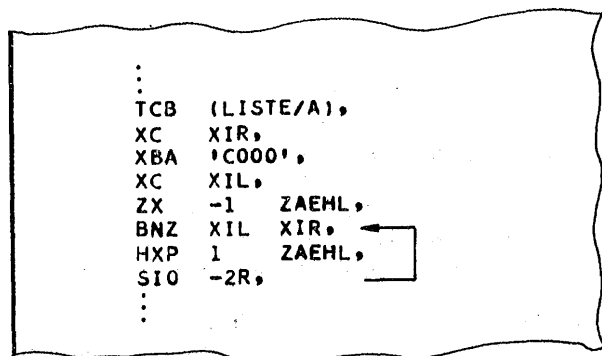


Bild 9.4 Beispiel zum Befehl BNZ

Im Bild 9.4 ist ein einfaches Beispiel mit dem Befehl BNZ gezeigt. Beim Ganzwort mit der Adresse LISTE beginnt eine Liste von 12-Bit-Zeichen (12 'C'). Sie werden in das Register A gebracht, und dort wird abgefragt, ob sie den Wert 0 haben. In der Indexzelle ZAEHL wird die Anzahl der zu Beginn der Liste stehenden Nullen gezählt.

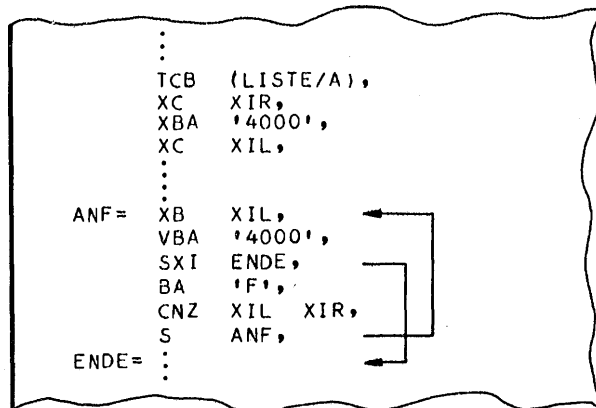


Bild 9.5 Beispiel zum Befehl CNZ

Im Bild 9.5 ist ein Beispiel mit dem Befehl CNZ aufgeführt. Es soll hier am Ende einer Abspeicherung von 4-Bit-Zeichen durch den Befehl CNZ das letzte Wort mit dem Zeichen F (\cong LLLL) aufgefüllt werden. Dazu wird die Indexzelle XIL abgefragt, ob die Zeichennummer Null ist. Da die Indexzelle außerdem die Länge der Zeichen enthält, muß die Länge erst subtrahiert werden. Es wird so oft das Zeichen F abgespeichert, bis die Zeichennummer 0 ist. Da der Befehl CNZ das Register A verändert, muß es jedesmal mit dem Befehl BA auf den Wert F gesetzt werden.

Die Befehle BNZ und CNZ arbeiten unabhängig von der Typenkennung. Bei dem Befehl BNZ erhält das Zeichen im Register A die gleiche Typenkennung wie das Wort im Speicher, in dem das Zeichen stand. Bei dem Befehl CNZ bleibt die Typenkennung des Wortes im Speicher erhalten.

Folgende Nebenwirkungen treten bei den Befehlen auf. Die Inhalte der Register D und Y sind undefiniert. Das Register B enthält den alten Wert der Indexzelle i_L . Bei dem Befehl CNZ ist außerdem im Register A der neue Inhalt der Speicherzelle, also mit den gerade abgespeicherten Zeichen. Im Register Q steht das abgespeicherte Zeichen an der Stelle, an der es in die Speicherzelle abgespeichert wurde. Die anderen Bits sind Null.

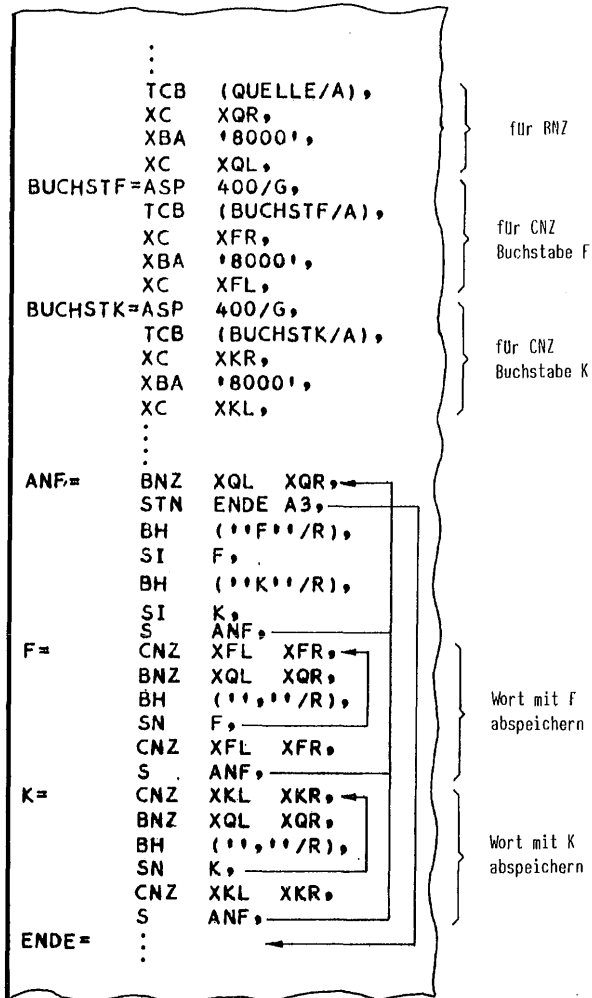


Bild 9.6 Beispiel Aussortieren der Namen, die mit dem Buchstaben F und K beginnen

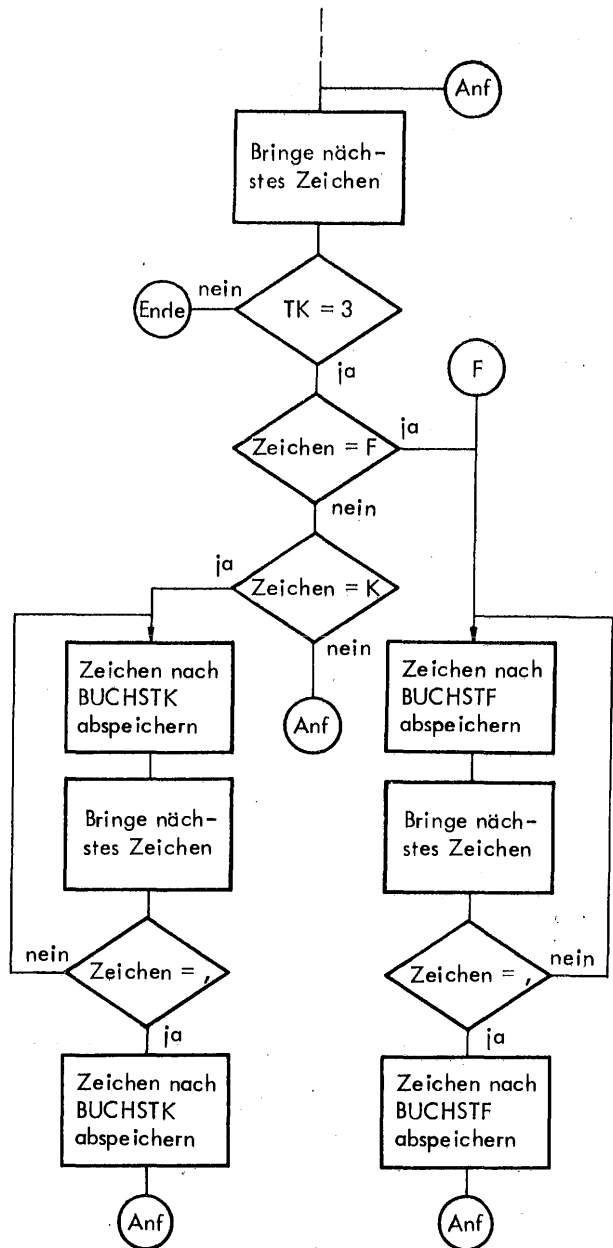


Bild 9.7 Ablaufdiagramm zum Beispiel Bild 9.6

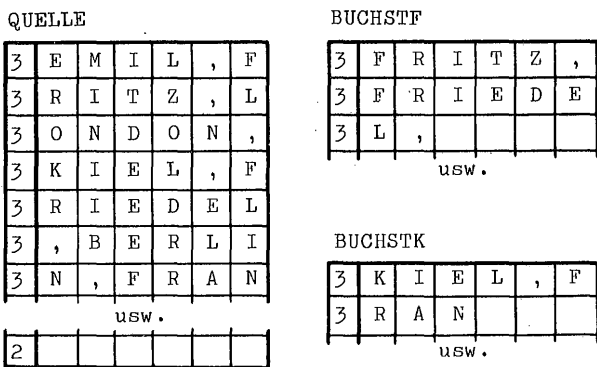


Bild 9.8 Speicherbereich zum Beispiel Bild 9.6

In den Bildern 9.6 bis 9.8 ist ein Beispiel für die Befehle BNZ und CNZ gezeigt. In einem Speicherbereich mit dem Namen QUELLE sind Namen abgespeichert. Jeder Name wird durch ein Komma abgeschlossen. Der Bereich wird durch ein Wort mit der Typenkennung 2 abgeschlossen. Aus den Namen werden jene, die mit dem Buchstaben F beginnen, in den Speicherbereich BUCHSTF gebracht und die Namen, die mit dem Buchstaben K anfangen, in den Speicherbereich BUCHSTK. Die Zeichen liegen im Zentralcode vor (8-Bit-Code).

9.2. Umschlüsseln

Um die Zeichen eines Codes in einen anderen Code umzuschlüsseln, steht der Befehl

```
US s i Umschlüsseln <A> := <A>umgeschlüsselt
```

- s₁: Bitlänge der Zeichen 6, 8 oder 12 (12 Bits)
- s₂: E = ein Zeichen rechtsbündig
G = alle Zeichen 8, 6 oder 4

zur Verfügung. Die Zeichen können eine Länge von 6, 8 oder 12 Bits haben. Die Länge des Zeichens wird als Spezifikation im Adressenteil des Befehls angegeben. Da kurze Zeichen durch Hinzufügen eines Null-Bits erweitert werden können, lassen sich alle Zeichenlängen bis zu 12 Bits umschlüsseln.

Im Adressenteil des Befehls kann angegeben werden, ob nur ein Zeichen (E) oder ob alle Zeichen (G) des Registers A umgeschlüsselt werden.

Für den Befehl muß eine Umschlüsseltabelle zur Verfügung stehen, deren Anfangsadresse in einer Indexzelle steht. Die Adresse dieser Indexzelle wird im Adressenteil des Befehls US angegeben. Der Aufbau der Umschlüsseltabelle ist im Abschnitt 9.2.3. beschrieben.

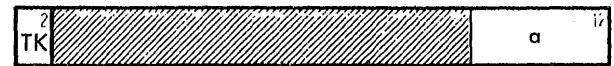
Der Befehl US ist unabhängig von der Typenkennung. Das Register A erhält immer die Typenkennung, die das Wort der Umschlüsseltabelle hat, in der das (umgeschlüsselte) Zeichen steht.

9.2.1. Einzelzeichen

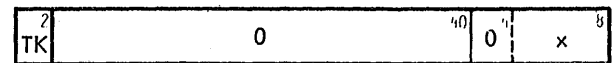
Wird im Adressenteil der Befehle die Spezifikation E angegeben, so wird nur ein Zeichen umgeschlüsselt. Es muß rechtsbündig im Register A stehen. Der Rest des Registers A ist bedeutungslos. Nach der Ausführung des Befehls steht das umgeschlüsselte Zeichen an Stelle des umzuschlüsselnden Zeichens, der Rest des Registers A wird auf Null gesetzt. Damit hat das Zeichen vor und nach der Umschlüsselung die gleiche Länge.

Im Bild 9.9 ist gezeigt, wie ein 12-Bit-Zeichen umgeschlüsselt wird in ein 8-Bit-Zeichen. Der Befehl US schlüsselt das 12-Bit-Zeichen stets in ein 12-Bit-Zeichen um. Ist der Code, in dem umgeschlüsselt wird, ein 8-Bit-Code, so sind die restlichen 4 Bits jedoch Null, und es werden nur die rechten 8 Bits weiterverarbeitet.

12-Bit-Code



umgeschlüsselt



8-Bit-Code

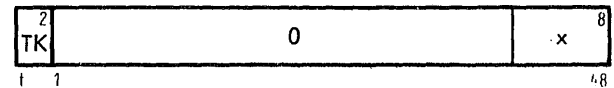
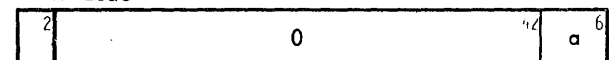


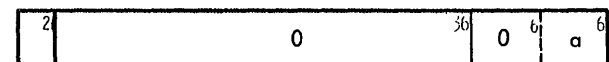
Bild 9.9 Umschlüsseln 12-Bit-Code in 8-Bit-Code

Im Bild 9.10 ist gezeigt, wie von einem 6-Bit-Code auf einen 12-Bit-Code umgeschlüsselt wird. Das 6-Bit-Zeichen wird um "Null"-Bits auf 12 Bits erweitert. Sein Wert wird dadurch nicht verändert. Dann wird ein 12-Bit-Code umgeschlüsselt.

6-Bit-Code



erweitert auf 12-Bit-Code



umgeschlüsselt

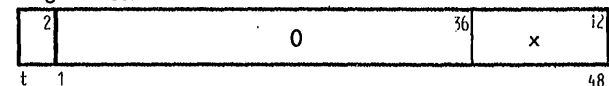


Bild 9.10 Umschlüsseln 6-Bit-Code in 12-Bit-Code

Als Nebenwirkung enthält das Register B nach der Ausführung die Adresse der Umschlüsseltabelle.

9.2.2. Alle Zeichen

Wird im Adressenteil des Befehls US die Spezifikation G (ganzes Wort) angegeben, so werden alle Zeichen des Registers A umgeschlüsselt. Je nach Länge der Zeichen (12, 8 oder 6 Bits) enthält das Register A dann 4, 6 oder 8 Zeichen.

Nach der Ausführung des Befehls stehen an Stelle der umzuschlüsselnden Zeichen die umgeschlüsselten Zeichen. Damit haben die Zeichen vor und nach der Umschlüsselung die gleiche Länge.

Soll, wie für die Umschlüsselung von Einzelzeichen an Hand der Bilder 9.9 und 9.10 gezeigt, die Zeichenlänge vor und nach der Umschlüsselung verschieden sein, so müssen die Zeichen mit geringerer Länge auf die größere Länge gebracht werden. Dies kann z.B. durch Schiften erreicht werden. Es ist dabei sinngemäß, wie in den Bildern 9.9 und 9.10 gezeigt, vorzugehen.

Als Nebenwirkung enthält das Register B nach der Ausführung die Adresse der Umschlüssel-tabelle. Des weiteren enthält das Register Q den gleichen Wert wie das Register A (nach der Umschlüsselung). Im Register Y steht die Anzahl der umgeschlüsselten Zeichen. Mit Hilfe der Befehle R und RLR kann auf das Register Y zurückgegriffen werden.

Im ersten Viertelwort dieser Tabelle, das ist das Viertelwort mit der Adresse 0 relativ zum Tabellenanfang, steht das Zeichen, in das das Zeichen mit dem Binärwert 0 umgeschlüsselt werden soll, in dem Viertelwort Nummer 1 (relative Adresse 1) das umzuschlüsselnde Zeichen für das Zeichen mit dem Binärwert 1 usw. Mit anderen Worten heißt dies, daß der Binärwert des umzuschlüsselnden Zeichens die Nummer des Viertelwortes (Adresse relativ zum Tabellenanfang) angibt, in dem das Zeichen steht, in das umgeschlüsselt werden soll. Im Bild 9.11 ist an einem Beispiel gezeigt, wie das Zeichen mit dem Binärwert 11 das Viertelwort adressiert, in dem das Zeichen "Fragezeichen" steht.

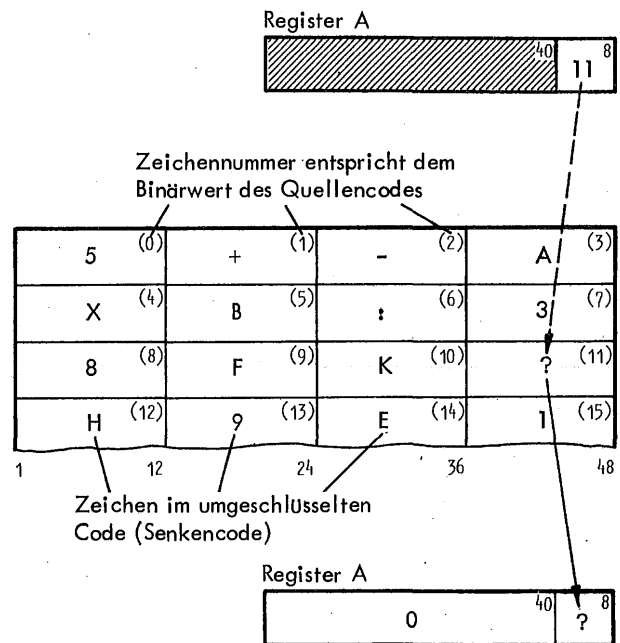


Bild 9.11 Beispiel für die Wirkung des Befehls US

9.2.3. Umschlüssel-tabelle

Für den Befehl US muß eine Umschlüssel-tabelle zur Verfügung stehen. Sie besteht aus Viertelwörtern (12 Bits). Die Tabelle kann mit einem Halbwort beginnen, und die Anfangsadresse der Tabelle muß in einer Indexzelle stehen. Die Adresse dieser Indexzelle wird im Adressenteil des Befehls US angegeben.

Auch wenn die umzuschlüsselnden Zeichen nur eine Länge von 8 oder 6 Bits haben, wird für jedes Zeichen ein Viertelwort benötigt. Die linken Bits werden dann auf Null gesetzt. Dementsprechend ist die Umschlüssel-tabelle bei 6, 8 bzw. 12 Bits langen Zeichen 64, 256 bzw. 4096 Viertelwörter lang.

Im Beispiel Bild 9.12 sind zwei Umschlüssel-tabelle für eine Umschlüsselung vom CCIT-Code in den Zentralcode angegeben. Durch die Buchstaben-Ziffern-Umschaltung waren zwei Tabellen mit 2⁵ Viertelwörtern erforderlich.

9.3. Beispiel

In den Bildern 9.12 und 9.13 ist an einem Beispiel gezeigt, wie mit Hilfe der Befehle BNZ, CNZ und US auf einfache Weise eine Zeichenfolge, die im für ALGOL modifizierten CCIT-Code Nummer 2 vorliegt, in den Zentralcode übersetzt werden kann. Die 5 Spuren des Lochstreifens liegen im 8-Bit-Code vor (die linken drei Bits sind 0). Für die Umschlüsseltabelle müssen 2^8 Viertelwörter vorgesehen werden. Davon sind je 32 Viertelwörter für die Ziffern- und Buchstaben-seite des Codes nötig.

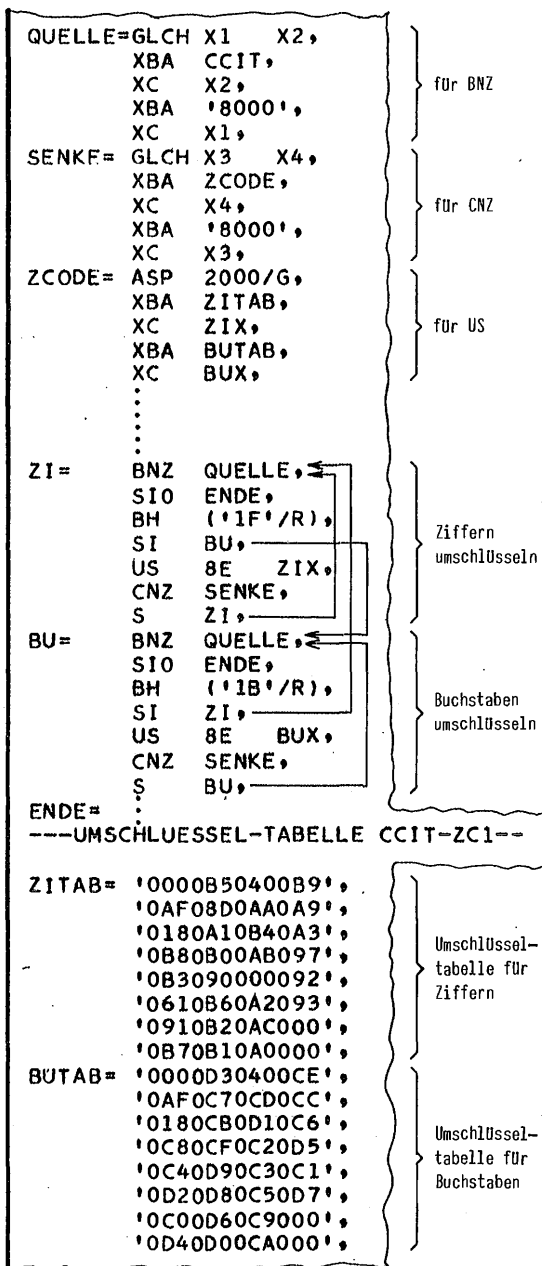


Bild 9.12 Beispiel Umschlüsseln von CCIT-2-Code in Zentralcode

Wie das Ablaufdiagramm zeigt, sind zwei getrennte Programmstücke vorgesehen, eins für die ziffernseitigen und eins für die buchstabenseitigen Zeichen. Tritt der Binärwert Bu auf, so wird auf den Bu-Zweig umgeschaltet, tritt Zi auf, so wird auf den Zi-Zweig umgeschaltet.

Die Schleife wird verlassen, wenn der nicht erlaubte Binärwert 0 auftritt (d.i. kein Loch).

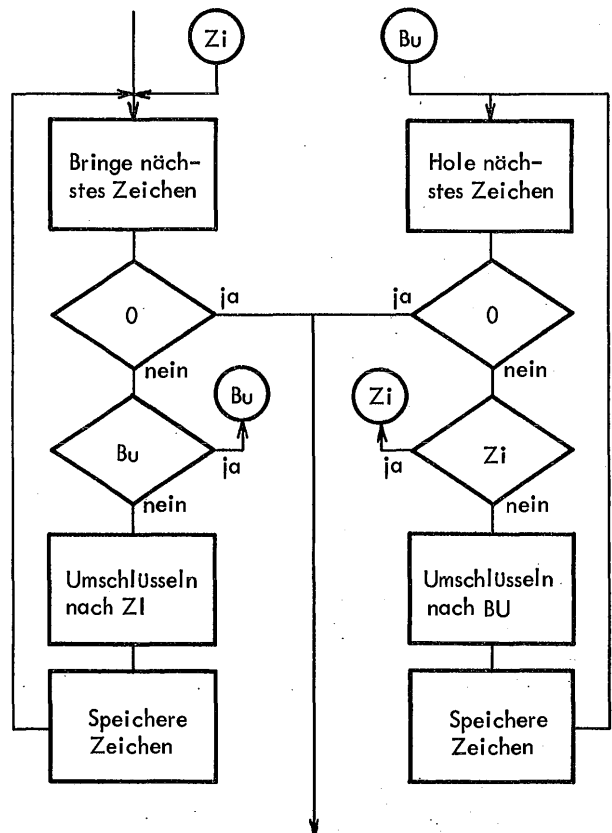


Bild 9.13 Umschlüsseln CCIT in Zentralcode

Umcodierungstabelle für ZI

CCIT		ZC1	
Dezimal	Zeichen	Zeichen	Sedezimal
0	*	IG	'00'
1	5	5	'B5'
2	W	WR	'40'
3	9	9	'B9'
4	V	V	'AF'
5	10	10	'8D'
6	,	,	'AA'
7	.	.	'A9'
8	Z	ZW	'18'
9))	'A1'
10	4	4	'B4'
11]]	'A3'
12	8	8	'B8'
13	0	0	'B0'
14	:	:	'AB'
15	=	=	'97'
16	3	3	'B3'
17	+	+	'90'
18	#	IG	'00'
19	x	x	'92'
20	'	'	'61'
21	6	6	'B6'
22	[['A2'
23	/	/	'93'
24	-	-	'91'
25	2	2	'B2'
26	;	;	'AC'
27	ZI	IG	'00'
28	7	7	'B7'
29	1	1	'B1'
30	(('A0'
31	BU	IG	'00'

Bild 9.14 Gegenüberstellung von CCIT-2-Code
ZC1-Code für Einstellung ZI

Umcodierungstabelle für BU

CCIT		ZC1	
Dezimal	Zeichen	Zeichen	Sedezimal
0	#	IG	'00'
1	T	T	'D3'
2	W	WR	'40'
3	O	O	'C8'
4	V	V	'AF'
5	H	H	'C7'
6	N	N	'CD'
7	M	M	'CC'
8	Z	ZW	'18'
9	L	L	'CB'
10	R	R	'D1'
11	G	G	'C6'
12	I	I	'C8'
13	P	P	'CF'
14	C	C	'C2'
15	V	V	'D5'
16	E	E	'C4'
17	Z	Z	'D9'
18	D	D	'C3'
19	B	B	'C1'
20	S	S	'D2'
21	Y	Y	'D8'
22	F	F	'C5'
23	X	X	'D7'
24	A	A	'C0'
25	W	W	'D6'
26	J	J	'C9'
27	ZI	IG	'00'
28	U	U	'D4'
29	Q	Q	'D0'
30	K	K	'CA'
31	BU	IG	'00'

Bild 9.15 Gegenüberstellung von CCIT-2-Code und
ZC1-Code für Einstellung BU

10. TEILWÖRTER

Mit Hilfe der Teilwortbefehle können Teile eines Ganzwortes verarbeitet werden. Zu diesem Zweck stehen folgende spezielle Befehle zur Verfügung. Der Befehl BT bringt ein Teilwort in das Register A. Mit den Befehlen AT und SBT können Teilwörter addiert bzw. voneinander subtrahiert werden. Mit dem Befehl CT können Teilwörter in den Speicher gebracht werden. Die Befehle sind unabhängig von der Typenkennung.

Welcher Teil eines Wortes verarbeitet wird, ist durch eine Maske festgelegt, die vor der Ausführung des Befehls im Register Q steht. Sie enthält gleichzeitig eine Angabe, um wieviel Stellen geschiftet wird.

Die nach der Ausführung der Teilwortbefehle verbleibenden Operanden können selbstverständlich mit anderen Befehlen weiterverarbeitet werden.

10.1. Maske

Bevor einer der Teilwortbefehle ausgeführt wird, muß im Register Q eine Maske bereitgestellt werden. Alle Binärstellen, bei denen im Register Q eine Null ist (0-Feld), bilden das Teilwort. Die anderen Binärstellen werden auf Null gesetzt (L-Feld der Maske). Das Nullfeld der Maske schneidet also das Teilwort aus dem Ganzwort heraus.

Des weiteren wird das Teilwort so weit nach rechts geschiftet, daß es zur Verarbeitung rechtsbündig im Register A steht. Bei der Abspeicherung mit dem Befehl CT wird das Teilwort in die gemäß Nullfeld der Maske angegebenen Stellen zurückgespeichert.

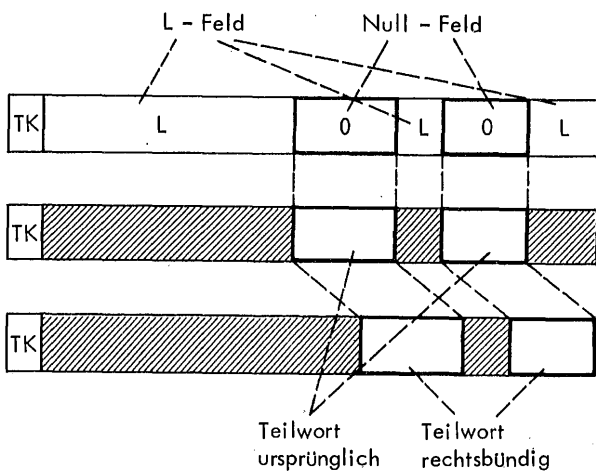


Bild 10.1 Maske und Teilwort

10.2. Teilwort bringen

Für das Bringen eines Teilwortes ist der Befehl

BT	n	bringe Teilwort	$\langle A \rangle_x := \langle n \rangle_x$ falls $\langle Q \rangle_x = 0$
			$\langle A \rangle_x := 0$ falls $\langle Q \rangle_x = 1$
			$\langle A \rangle := \langle A \rangle$ geschiftet

$x: 1, 2, \dots, 48$

vorhanden. Er adressiert ein Ganzwort und bringt aus diesem Ganzwort alle die Binärstellen, bei denen im Register Q eine Null steht (Null-Feld der Maske), in das Register A. Die anderen Binärstellen im Register A werden auf 0 gesetzt.

Des weiteren wird der Inhalt des Registers A um soviel Stellen nach rechts geschiftet, wie im Register Q rechtsbündig aufeinanderfolgende L-Bits sind. Dies bedeutet, daß das Teilwort rechtsbündig im Register A steht.

Das Register A erhält die gleiche Typenkennung wie die Speicherzelle.

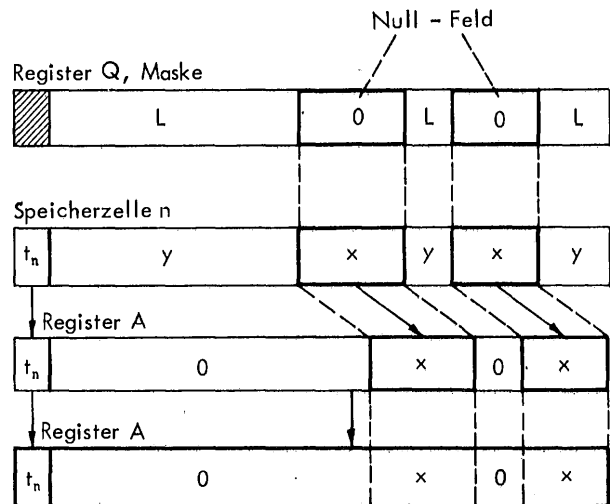


Bild 10.2 Wirkung des Befehls BT

L

IAS - Handbuch

Dez. 68

10.3. Addition - Subtraktion

Die mit dem Befehl BT gebrachten Teilwörter können von beliebigen Befehlen gemäß deren Regeln verarbeitet werden, also auch durch Additions- und Subtraktionsbefehle. Für die Teilwörter gibt es jedoch zwei spezielle Befehle.

AI	n	Addiere Teilwort	$\langle A \rangle := \langle A \rangle_x + \langle n \rangle_x$
SBT	n	Subtrahiere Teilwort	$\langle A \rangle := \langle A \rangle_x - \langle n \rangle_x$

für $\langle 0 \rangle_y = 0$

Der erste Operand für beide Befehle steht im Register A. Der zweite Operand wird aus dem Speicher geholt, und zwar auf die gleiche Weise, wie bei dem Befehl BT.

Die beiden Operanden werden addiert bzw. subtrahiert, aber nur in den Binärstellen, die das Teilwort bilden. Dabei wird, unabhängig von der Typenkennung, so verfahren, als ob die dem Null-Feld der Maske entsprechenden Binärstellen zusammenhängen und eine ganze, positive Festkommazahl enthielten. Negative Zahlen sind also nicht möglich. Bild 10.3 veranschaulicht dies.

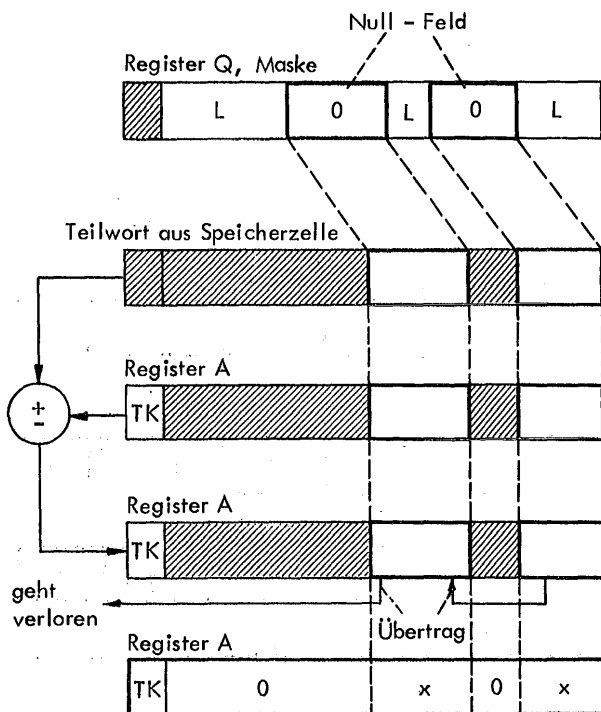


Bild 10.3 Wirkung des Befehls AT bzw. SBT

Es sei noch besonders darauf hingewiesen, daß die Maske im Register Q zum Bilden des Teilwortes verwendet wird und nur die Teilwörter addiert bzw. subtrahiert werden. Es sind also die Teilwörter nach rechts geschiftet, wie bei dem Befehl BT beschrieben, die Maske jedoch nicht. Es werden also nicht die gleichen Binärstellen addiert, wie das Null-Feld der Maske angibt.

Wird bei der Addition das Ergebnis größer als es das Teilwort erlaubt oder wird bei der Subtraktion das Ergebnis kleiner als Null, so wird ein BÜ-Alarm gegeben.

Als Nebenwirkung wird bei beiden Befehlen der Inhalt des Registers Q (Maske) auch in das Register D gebracht.

In den Bildern 10.4 und 10.5 ist das Prinzip gezeigt, wie die Addition bzw. die Subtraktion bei zwei Teilwörtern intern durchgeführt wird. Bei der Addition geht ein links aus dem Register herauslaufendes Bit verloren, und es wird ein BÜ-Alarm gegeben. Er zeigt an, daß das Ergebnis übergelaufen ist. Das Ergebnis ist dann, über alle 48 Binärstellen betrachtet, um 2^{48} zu klein. Das Teilwort aus x Binärstellen ist dann um 2^x zu klein.

Bei der Subtraktion wird der Subtrahend invertiert (B-1-Komplement) und dann addiert. Der notwendige Einerrücklauf wird durch Addieren einer 1 berücksichtigt (er tritt bei SBI immer auf). Ist das links vom Bit Nummer 1 gedachte Bit 0, so wird BÜ-Alarm gegeben. Dieses Ergebnis ist dann, über alle 48 Bits gesehen, um 2^{48} zu groß. Das Teilwort aus x Binärstellen ist dann um 2^x zu groß.

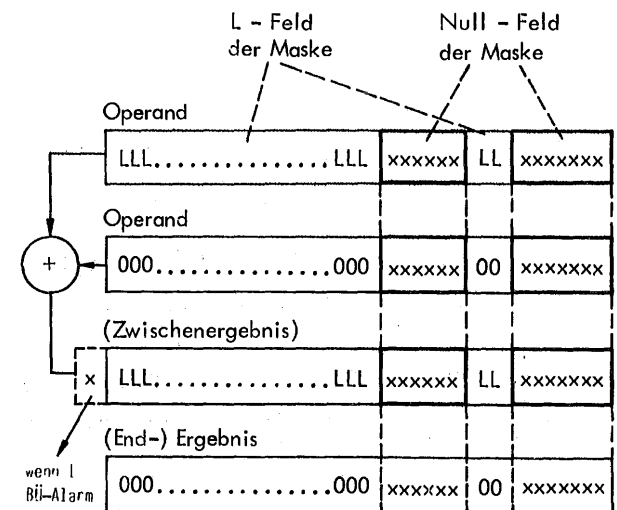


Bild 10.4 Prinzip des internen Ablaufs des Befehls AT

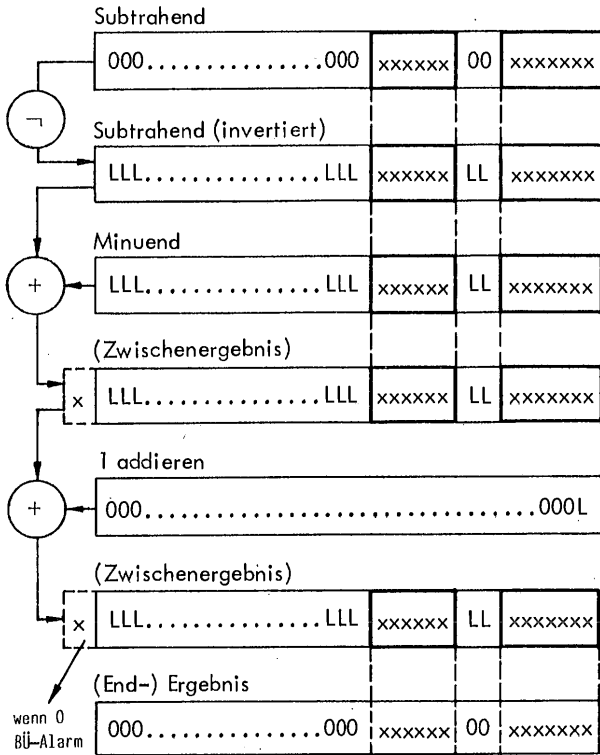


Bild 10.5 Prinzip des internen Ablaufs des Befehls SBT

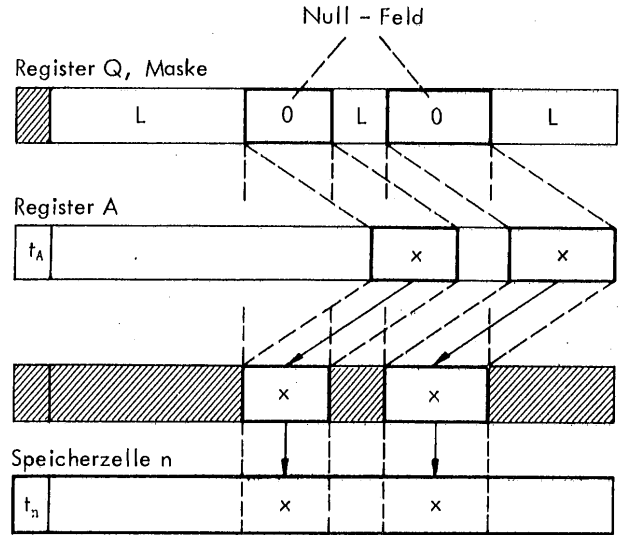


Bild 10.6 Wirkung des Befehls CT

10.4. Teilwort speichern

Um ein Teilwort abzuspeichern, steht der Befehl

```
CT n Speichere Teilwort <n>x := <A>x
```

x: 1,2,...,48

zur Verfügung. Im Register Q muß dazu eine Maske stehen. Sie gibt an, an welche Binärstellen der Speicherzelle das Teilwort gespeichert werden soll.

Der Befehl CT ist das Gegenstück zum Befehl BT. Dort wurde gemäß Null-Feld der Maske ein Teilwort aus dem Speicher geholt und um soviel Stellen nach rechts geschiftet, wie rechtsbündig im Register Q (Maske) aufeinanderfolgende L-Bits stehen. Entsprechend wird bei dem Befehl CT erst nach links geschiftet, und dann werden die dem Null-Feld der Maske entsprechenden Binärstellen in die gleichen Binärstellen der Speicherzelle eingesetzt. Die anderen Bits der Speicherzelle und die Typenkennung bleiben unverändert. Das Register A bleibt ebenfalls unverändert.

Als Nebenwirkung wird der neue Inhalt der Speicherzelle in das Register D gebracht.

10.5. Beispiele

Im Beispiel Bild 10.7 ist gezeigt, wie in einer Liste mit dem Namen QUELLE aus den Ganzwörtern die Binärstellen 13 bis 24 als Teilwort in das Register A gebracht werden. Die Maske wird mit dem Befehl BQ, der im Adressenteil die Maske als Literal hat, in das Register Q gebracht. Es wird das erste Wort gesucht, bei dem der Wert in den Binärstellen 13 bis 24 größer als 100 ist.

Liste QUELLE

3			
3			
3			
3			
3			
t	1	12	24

```

:
BQ ('FFF000FFFFFF'),
BH (100/3),
ZX -2 ZAEHL,
SCHL = HXP 2 ZAEHL,
M ZAEHL,
BT QUELLE,
SKG SCHL,
:

```

Bild 10.7 Beispiel zum Befehl BT

TELEFUNKEN
COMPUTER

TAS - HANDBUCH

Adressenrechnung

M

INHALT



<u>ADRESSENRECHNUNG</u>	0 - 1
<u>1. AUFBAU DER ADRESSEN UND INDEXGRÖSSEN</u>	1 - 1
1.1. Bereich	1 - 1
1.2. Vorzeichen	1 - 2
1.3. BereichsÜberschreitung	1 - 2
1.4. Vergleiche	1 - 2
1.5. Indexspeicher	1 - 2
1.6. Adressen bei Sprungbefehlen	1 - 3
<u>2. SETZEN UND LÖSCHEN</u>	2 - 1
<u>3. TRANSPORTE</u>	3 - 1
3.1. Register B - Indexzelle	3 - 1
3.2. Register B - Speicherzelle	3 - 1
3.3. Indexzelle - Rechenwerksregister	3 - 1
3.4. Indexzelle - Indexzelle	3 - 2
<u>4. ARITHMETISCHE INDEXOPERATIONEN</u>	4 - 1
4.1. Register B und Adressenteil	4 - 1
4.2. Register B und Speicherzelle	4 - 1
4.3. Register B und Indexzelle	4 - 1
4.4. Register und Indexzelle.....	4 - 2
4.5. Indexzelle und Parameter	4 - 2
4.6. Indexzelle und Indexzelle	4 - 2
4.7. Indexzelle und Register	4 - 2
4.8. Multiplizieren	4 - 2
<u>5. ABFRAGEN - VERZWEIGUNGEN - SPRÜNGE</u>	5 - 1
<u>6. SCHIFTEN</u>	6 - 1
<u>7. MODIFIZIEREN</u>	7 - 1
7.1. Modifizierung 1. Art	7 - 3
7.2. Modifizierung 2. Art	7 - 3
<u>8. MODIFIZIERBEFEHLE</u>	8 - 1
8.1. Modifikator aus Indexzelle	8 - 1
8.2. Modifikator aus Adressenteil	8 - 2
8.3. Modifikator aus Speicher	8 - 3
<u>9. ERSETZEN</u>	9 - 1
<u>10. ERSETZBEFEHLE</u>	10 - 1
10.1. Adressenteil aus Indexzelle	10 - 1
10.2. Adressenteil aus Register B	10 - 3
10.3. Adressenteil über Unterprogrammregister	10 - 4
10.4. Adressenteil relativ zum Befehl	10 - 5
10.5. Operand aus Register	10 - 5
10.6. Befehl T	10 - 7



11.	<u>AUFBAU VON SCHLEIFEN</u>	11 - 1
11.1.	Zählgröße	11 - 1
11.2.	Zählgröße gleichzeitig Modifiziergröße	11 - 2
11.3.	Rechenwerksregister	11 - 3
11.4.	Merklichter	11 - 6

Die Adressenrechnung wird innerhalb des Befehlswerkes durchgeführt. Es hat dazu ein rechenfähiges Register – das Register B – in dem addiert und subtrahiert werden kann. Des weiteren kann in diesem Register geschiftet werden. Das bedeutet, daß auch mit einem Faktor 2^p multipliziert bzw. dividiert werden kann.

Des weiteren kann der Inhalt des Registers B mit Null verglichen und in Abhängigkeit davon gesprungen werden. Dies ergibt die Möglichkeit von Verzweigungen.

Das Register B arbeitet äußerst eng mit dem Indexspeicher zusammen. Der Indexspeicher hat 256 Zellen, die durch eine Adresse von 8 Bits adressiert werden. Jede der Zellen hat eine Länge von 24 Bits und kann damit jede Kernspeicheradresse aufnehmen. Eine Erweiterung des Indexspeichers ist jederzeit möglich.

Die vier zuletzt benutzten Indexzellen sind immer im Befehlswerk. Sie sind in 4 assoziativen Registern gespeichert. Für diese 4 Indexgrößen entfällt die Zugriffszeit, d.h. sie stehen ohne zusätzliche Zeit zur Verfügung, was die Rechenzeit bei oft durchlaufenen Schleifen wesentlich verkürzt. Erst wenn weitere Indexzellen benötigt werden, wird ein Speicherzugriff erforderlich. Dabei wird das am längsten nicht benutzte Indexregister überschrieben. Dieser Austausch der Indexgrößen geht automatisch vor sich, so daß dafür vom Programmierer keine zusätzliche Arbeit verlangt wird, ihm aber den Vorteil der Rechenzeitverkürzung bringt.

Die Adressenrechnung wird im wesentlichen für die Bildung von Schleifen im Programm benötigt. Das Prinzip ist im Bild 1 gezeigt.

Die Indexgrößen, die innerhalb der Schleife benötigt werden, sind auf einen definierten Wert zu setzen. Bei den Befehlen, die innerhalb der Schleifen auftreten, werden ggf. die Adressen mit Hilfe von Modifizier- oder Ersetzbefehlen über die Indexgrößen verändert (hochgezählt). Anschließend sind die Indexgrößen hochzuzählen.

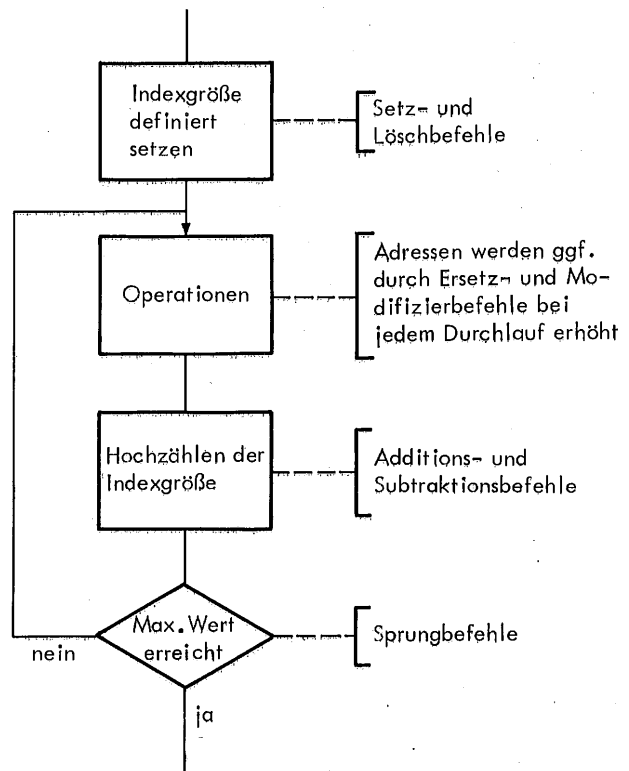


Bild 1 Prinzip der Schleifenbildung

Bei den weiteren Durchläufen wird dann mit den hochgezählten Indexgrößen gearbeitet. Schließlich wird verglichen, ob die Indexgröße den gewünschten Maximalwert erreicht hat. Ist er erreicht, so wird im Programm fortgefahren. Im anderen Fall wird die Schleife erneut durchlaufen.



TAS - Handbuch

Dez. 68

1. AUFBAU DER ADRESSEN- UND INDEXGRÖSSEN

Bei den hier beschriebenen Adressen handelt es sich um die Adressen des Kernspeichers. Diese Adressen können stets in Indexzellen abgespeichert werden. Jedoch können die Indexzellen außerdem auch Zählgrößen enthalten, die zur Veränderung (modifizieren) von Adressen verwendet werden oder als Abbruchkriterien für Schleifen dienen. Des weiteren werden die Indexzellen zur Abspeicherung aller, die Adressenrechnung betreffenden Konstanten und Zwischenergebnisse verwendet.

1.1. Bereich

Die Speicheradressen können bei der Niederschrift eine Länge von 16 Bits annehmen. Damit können 65 536 Halb- wörter oder 32 768 Ganzwörter adressiert werden. Bei den Befehlen, die einen Operanden aus dem Speicher holen, bzw. in den Speicher bringen, liegt das adressierte Wort in der Großseite 0 (K- oder V-Bereich). Bei Sprungbe- fehlen weist die Adresse auf einen Befehl, der in der Großseite 1 liegt.

In jedem Fall kann bei der Niederschrift nur eine Adresse angegeben werden, die maximal 16 Bits hat. Auch im Speicher steht das Programm stets mit einer Adresslänge von 16 Bits.

Zur Ausführung wird Befehl für Befehl in das Befehlswerk geholt. Dies geschieht in der Abrufphase. Innerhalb der Abrufphase wird der Adressenteil jedes Befehls links um 8 Null-Bits ergänzt, so daß der Adressenteil nunmehr 24 Bits lang ist.

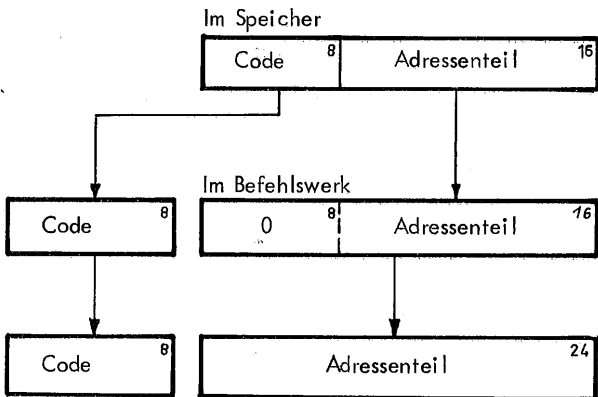


Bild 1.1 Erweiterung des Adressenteils auf 24 Bits

Der Inhalt des Adressenteils hat sich damit jedoch nicht verändert. Mit Hilfe von Modifizierbefehlen kann nun der Adressenteil verändert werden, indem eine Größe - die Modifiziergröße (der Modifikator) - addiert bzw. subtrahiert wird. Nunmehr kann also eine Adresse auch größer werden als 16 Bits und damit der gesamte Speicher adressiert werden.

Bei dem 24 Bits langen Adressenteil wird das 1. Bit bei Addition und Subtraktion sowie bei Vergleichen stets als Vorzeichen betrachtet. Somit ist es möglich, mit negativen

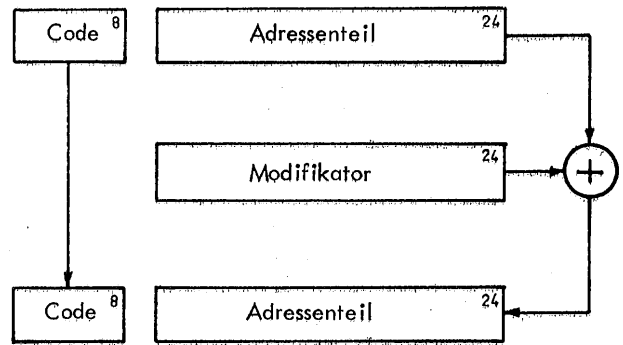


Bild 1.2 Änderung des Adressenteils durch Modifikator

tiven Größen zu rechnen. Dies ist bei Adressen nur für Zwischenergebnisse von Interesse. Die Reihenfolge von Addition und Subtraktion ist damit beliebig (siehe Bild 1.3).

o	OLOL	≙	+5	Ausgangswert
+	L	OLLL	≙	-8
<hr/>				
L	LL00	≙	-3	negatives Zwischenergebnis
+	o	L0LO	≙	+10
<hr/>				
o	OLLL	≙	+7	Endwert (positiv)

Bild 1.3 Beispiel für negativen Adressenteil

Zur Adressierung einer Speicherzelle werden nur die rechten 22 Bits des Adressenteils verwendet. Die linken beiden Bits (einschl. Vorzeichen) bleiben unberücksichtigt. Eine Adresse, die dem Speicher übergeben wird, muß also im Adressenteil stets positiv sein.

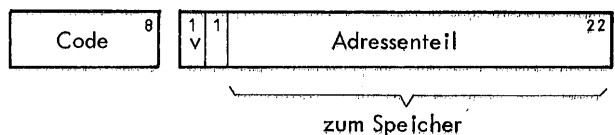


Bild 1.4 Übergabe einer Adresse an den Speicher

Damit ist die maximal erreichbare Adresse $2^{22}-1 = 4\,194\,303$, das sind über 4 Millionen Halb- wörter bzw. 2 Millionen Ganzwörter. Die maximale Adresse ist jedoch durch den Speicherausbau begrenzt (abzüglich des vom Betriebssystem belegten Speicherplatzes). Diese Grenze dürfte jedoch nur bei sehr großen Programmen erreicht werden.

Ein Programm, das diese Grenze überschreitet, kann geteilt werden. Jedes Teilstück darf die Grenze nicht überschreiten. Es ist dann beim Lauf des Programms nur jeweils ein Teilstück im Kernspeicher, während die anderen Teilstücke im Hintergrundspeicher sind. Näheres darüber wird an anderer Stelle gesagt.

1.2. Vorzeichen

Bei den Additions- und Subtraktionsbefehlen sowie bei den Vergleichen (Sprungbefehlen) wird das 1. Bit (linkes Bit) als Vorzeichen gewertet. Ist das 1. Bit = 0, so ist die Zahl positiv, ist es 1, so ist die Zahl negativ.

Negative Zahlen werden stets durch das (B-1)-Komplement dargestellt, d.h. bei einer negativen Zahl sind gegenüber der positiven Zahl alle Bits invertiert.

$$\begin{array}{r} 0 \text{ L } 0 \text{ L } 0 \quad \hat{=} \quad +10 \\ \text{L } 0 \text{ L } 0 \text{ L} \quad \hat{=} \quad -10 \end{array}$$

In diesem Sinne gibt es also sowohl eine positive als auch eine negative Null.

$$\begin{array}{r} 0 \text{ } 0 \text{ } 0 \text{ } 0 \quad \hat{=} \quad +0 \\ \text{L } \text{L } \text{L } \text{L} \quad \hat{=} \quad -0 \end{array}$$

Bei einigen Befehlen muß dies beachtet werden. Wird z.B. eine negative Null um eine Stelle nach rechts geschiftet, so entsteht die Zahl $+(2^{23}-1)$. Bei den arithmetischen Operationen und bei den Vergleichen sind negative und positive Null gleichwertig.

Eine Subtraktion wird so durchgeführt, daß die Zahl, die subtrahiert werden soll, in allen Binärstellen invertiert wird. Damit hat sie ihr Vorzeichen gewechselt. Danach wird addiert.

Bei einigen Befehlen ist es auch möglich, den Inhalt des Registers B bzw. den Inhalt einer Indezzelle als vorzeichenlose Zahl aufzufassen. In diesem Falle kann die Zahl maximal den Wert $2^{24}-1$ annehmen. Es ist jedoch zu beachten, daß eine Reihe von Befehlen die Zahl als negativ betrachten, sobald sie den Wert von $2^{23}-1$ überschreitet. Steht z. B. im Register B die Zahl 2^{23} , und wird sie mit dem Befehl TXR in das Register A gebracht, so steht dort die Zahl $-(2^{23}-1)$, sie hat also ihren Wert verändert (siehe Bild 1.5).

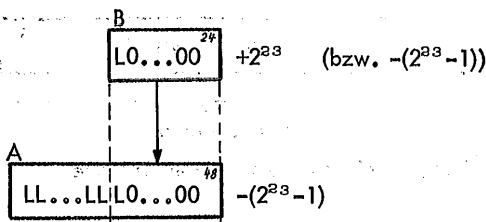


Bild 1.5 Beispiel mit Befehl TXR

1.3. Bereichsüberschreitung

Bei Addition und Subtraktion wird stets ein Bitmuster verknüpft. Ein links herauslaufendes L-Bit wird rechts hinzuaddiert (Einerrücklauf).

Wird das 1. Bit als Vorzeichen angesehen (in den meisten Fällen), ist die maximal darstellbare Zahl $\pm(2^{23}-1)$. Wird diese Zahl um 1 überschritten, so wird das Vorzeichen gewechselt.

$$\begin{array}{r} 0 \text{ LL...LL} \quad \hat{=} \quad +(2^{23}-1) \\ + 0 \text{ } 00...0\text{L} \quad \hat{=} \quad +1 \\ \hline \text{L } 00...00 \quad \hat{=} \quad -(2^{23}-1) \end{array}$$

$$\begin{array}{r} \text{L } 00...00 \quad \hat{=} \quad -(2^{23}-1) \\ + \text{L } \text{LL...L}0 \quad \hat{=} \quad -1 \\ \hline 0 \text{ LL...LL} \quad \hat{=} \quad +(2^{23}-1) \end{array}$$

Bild 1.6 Überlauf bzw. Unterlauf im Register B

Dies bedeutet, daß bei einer Bereichsüberschreitung das Ergebnis um $2^{24}-1$ zu klein und bei einer Bereichsunterschreitung um den Wert zu groß ist. Eine Fehlermeldung wird nicht gegeben.

1.4. Vergleiche

Der Inhalt des Registers B kann mit der Zahl Null verglichen werden, und in Abhängigkeit vom Ergebnis des Vergleichs wird gesprungen oder nicht. Hierzu dienen die Sprungbefehle; sie ermöglichen Programmverzweigungen. Näheres siehe Abschnitt 5.

1.5. Indexspeicher

Die 256 Indezzellen, die jedem Programm zur Verfügung stehen, liegen innerhalb des übrigen Speicherbereiches. Es muß ein veränderlicher Speicherbereich sein. Seine Lage ist festgelegt, durch die Indexbasisadresse, die im Register X steht. Das Register X braucht vom Programmierer nicht besetzt werden. Es kann aber vom Programm mit den Befehlen ZI und BCI verändert werden. Das Register X enthält die Adresse, die die Indezzelle mit der Adresse 0 im Speicher (Halbwort) hat.

Im Beispiel Bild 1.7 werden 12 Indezzellen benötigt. Sie liegen im Speicher, beginnend bei der Adresse $n+1$. Diese Adresse steht im Register X. Die Speicherzelle $n+13$ wird bereits wieder als Speicheradresse verwendet. Die Indezzellen sind mit ihren Adressen 0 bis 11 (oder symbolisch) adressierbar.

Für den Programmierer sind die Indezzellen über ihre (Index-) Adressen 0 bis 255, bzw. durch ihre symbolischen Adressen ansprechbar. Sie sind auch über die Speicheradressen ansprechbar, wenn die Adressen bekannt sind.

Im Speicher werden für den Indexspeicher immer nur soviel Halbwörter belegt, wie Indexzellen benötigt werden.

Die Indexbasis wird vom Assembler gesetzt, so daß der Programmierer sich um diese Organisation nicht zu bemühen braucht. Er hat jedoch die Möglichkeit, die Indexbasis auf einen anderen Wert zu setzen. Damit kann man weitere Indexbereiche aufmachen und von einem Indexbereich auf den anderen umschalten.

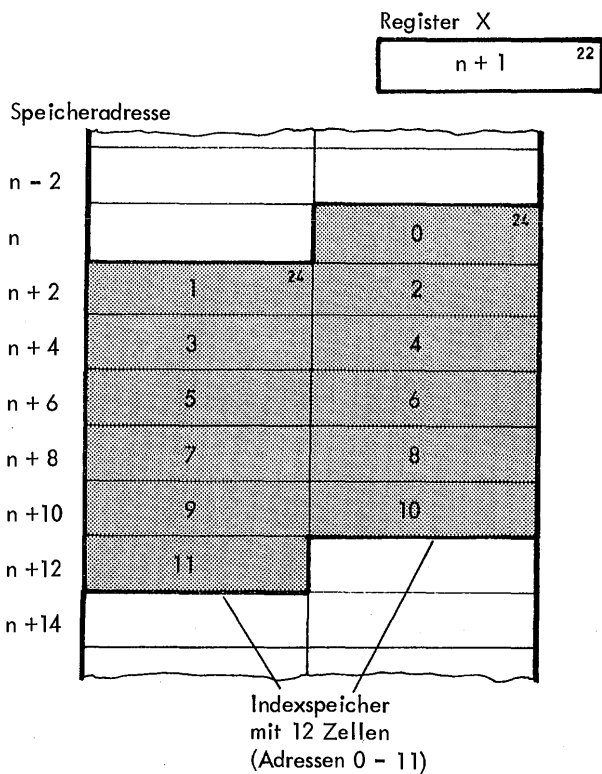


Bild 1.7 Beispiel für Indexspeicher

1.6. Adressen bei Sprungbefehlen

Das Register F im Befehlswerk enthält die Adresse des Befehls, der zur Zeit bearbeitet wird. Sobald ein Befehl ausgeführt ist, wird das Register F erhöht und zeigt auf den nächsten Befehl. Die Befehle werden also mit Hilfe des Registers F sequentiell abgearbeitet.

Zur Vereinfachung wurde die vorstehend beschriebene Art für alle die Programmierung betreffenden Beschreibungen gewählt, d.h. das Befehlsregister F wird am Schluß der Ausführungsphase um 1 erhöht. Intern im Rechner wird jedoch das Register bereits in der Abrufphase um 1 erhöht.

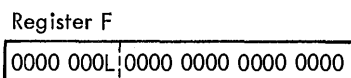
Durch die Sprungbefehle kann die aufsteigende Reihenfolge unterbrochen werden, um an einer anderen Stelle im Programm fortzufahren (Programmverzweigung). Dies geschieht dadurch, daß das Register F auf einen anderen Wert gesetzt wird und dort dann mit dem sequentiellen Abruf der Befehle fortfährt.

Alle Befehle werden vom Assembler in die Großseite 1 gelegt. Damit liegt der erste Befehl bei der Adresse 65 636, das ist binär

L 0000 0000 0000 0000

Diese Adresse hat also 17 Bits, und die Adressen aller folgenden Befehle haben ebenfalls mindestens 17 Bits.

Der Assembler setzt nun das Register F auf den Wert 65 536. Damit beginnt der Programmablauf mit dem ersten Befehl. Das Register steht also auf dem Wert



Bei allen Sprungbefehlen wird nun die Sprungadresse relativ zu dieser im Register F stehenden Adresse aufgefaßt. Da der Adressenteil der Befehle 16 Bits lang ist, kann damit die ganze Großseite 1 erreicht werden.

Bei einem Sprungbefehl wird, wie bei allen Befehlen, innerhalb der Abrufphase der 16 Bits lange Adressenteil durch 0-Bits auf 24 Bits verlängert. Die Wirkung eines Sprunges besteht darin, daß die Sprungadresse in das Register F gebracht wird. Im Normalfall werden jedoch bei den meisten Sprungbefehlen nur die rechten 16 Bits in das Register F übertragen. Die linken 8 Bits bleiben erhalten. Damit kann nur innerhalb der Großseite gesprungen werden, die durch die linken 8 Bits im Register F adressiert ist - in den meisten Fällen die Großseite 1.

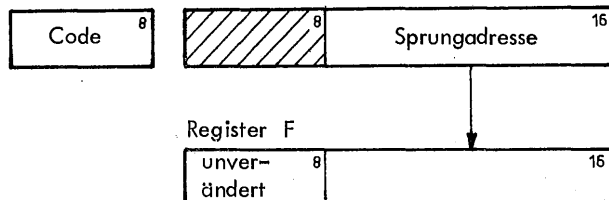


Bild 1.8 Wirkung der meisten Sprungbefehle

Nur die Befehle SE, SUE und alle Sprungbefehle, wenn sie mit MABI oder MU modifiziert sind, bilden eine Ausnahme. In diesen Fällen werden alle 24 Bits ins Register F gebracht, und es kann in eine andere Großseite gesprungen werden. Nach einem Sprung in eine andere Großseite wird mit den folgenden Sprungbefehlen innerhalb dieser neuen Großseite gesprungen.

Vorsicht: Soll nicht in eine andere Großseite gesprungen werden, so ist bei den vorstehenden Befehlen, die diese Ausnahme bilden, die richtige Adresse in 24 Bits mit Großseitenangabe anzugeben.

3. TRANSPORTE

Hier sind die Transporte aufgeführt, die das Register B und die Indexzellen betreffen. Es gibt folgende Transportmöglichkeiten:

- Register B - Indexzelle
- Register B - Speicherzelle
- Indexzelle - Rechenwerksregister
- Indexzelle - Indexzelle

Es werden immer 24-Bit-Größen transportiert. Bei den Rechenwerksregistern sind dies die rechten 24 Bits.

3.1. Register B - Indexzelle

Für Transporte zwischen dem Register B und einer Indexzelle sind die Befehle XB, XC und XCN vorhanden.

XB	i	Index: <u>Br</u> inge	$\langle B \rangle := \langle i \rangle$
XC	i	Index: <u>Spei</u> chern	$\langle i \rangle := \langle B \rangle$
XCN	i	Index: <u>Spei</u> chern <u>negativ</u>	$\langle i \rangle := -\langle B \rangle$

Beim Befehl XB wird der Inhalt einer Indexzelle in das Register B gebracht. Der Befehl XC speichert den Inhalt des Registers B in eine Indexzelle; beim Befehl XCN wird der Inhalt des Registers B mit umgekehrten Vorzeichen (alle Binärstellen werden invertiert) in die Speicherzelle gebracht.

3.2. Register B - Speicherzelle

Für den Transport zwischen Register B und einer Speicherzelle sind die Befehle TBC und TCB vorhanden.

TBC	m	Transport aus Register <u>B</u> nach <u>Spei</u> cher	$\langle m \rangle := \langle B \rangle$
TCB	m	Transport aus <u>Spei</u> cher nach Register <u>B</u>	$\langle B \rangle := \langle m \rangle$

Der Befehl TCB bringt den Inhalt des mit m adressierten Halbwortes unabhängig von der Typenkennung unverändert ins Register B. Dieser Befehl kann z.B. dazu verwendet werden, eine Indexzelle mit einer Konstanten zu besetzen, die größer als 16 Bits ist. Der Befehl kann auf 2. Art modifiziert werden.

Beim Befehl TBC wird der Inhalt des Registers B unverändert in das mit m adressierte Halbwort gebracht. Die Typenkennung der Speicherzelle bleibt unverändert.

DBER=	ASP	100/GD,
	TCB	(DBER/A),
	XC	X1,
VBER=	ASP	10/G,
	XB	X1,
	TBC	VBER+5,
	XB	X1,
	MCFU	(DBER/A),
	TBC	5,

Bild 3.1 Beispiele zu den Befehlen TBC und TCB

3.3. Indexzelle - Rechenwerksregister

Mit den Befehlen TRX und TXR kann ein Transport zwischen einer Indexzelle und dem angegebenen Rechenwerksregister durchgeführt werden.

TRX	s i	Transport aus <u>Rechenwerk</u> in <u>Indexzelle</u>	$\langle i \rangle := \pm \langle s_1 \rangle_{25-48}$ $\langle B \rangle := \pm \langle s_1 \rangle_{25-48}$
TXR	s i	Transport aus <u>Indexzelle</u> nach <u>Rechenwerk</u>	$\langle s_1 \rangle := \pm \langle i \rangle$ $\langle B \rangle := \pm \langle i \rangle$

s_1 : Register A, Q, D, H oder leer
 s_2 : leer: + } statt ±
 N: - }

Beim Befehl TRX wird die rechte Hälfte des Registers A, Q, D oder H in die mit i angegebene Indexzelle gebracht. Gleichzeitig steht diese Größe im Register B. Wird kein Register angegeben, so werden die Indexzelle und das Register B auf + 0 (bei Angabe von N auf -0) gesetzt.

Beim Befehl TXR wird der Inhalt der mit i angegebenen Indexzelle in die rechte Hälfte des angegebenen Registers gebracht. Das erste Bit der Indexgröße ist das Vorzeichen. Die linke Hälfte des Rechenwerksregisters wird vorzeichenrichtig aufgefüllt, d.h. die Indexgröße steht vorzeichenrichtig im Rechenwerksregister. Die Typenkennung wird auf 1 gesetzt. Es ist möglich, beim Befehl TXR (im Gegensatz zum Befehl TRX) mehrere Register anzugeben. Sie werden alle auf den gleichen Wert gesetzt. Gleichzeitig steht die Indexgröße im Register B.

Bei beiden Befehlen kann die Spezifikation N angegeben werden. Es wird dann die Größe mit umgekehrten Vorzeichen transportiert (alle Binärstellen werden invertiert).



TAS - Handbuch

Dez. 68

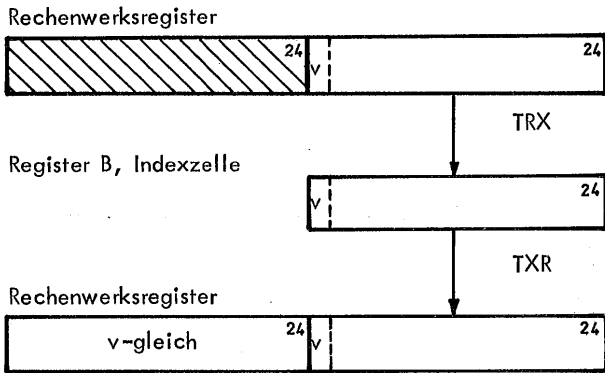


Bild 3.2 Wirkung von TRX und TXR

3.4. Indexzelle - Indexzelle

Mit dem Befehl TXX kann der Inhalt einer Indexzelle in eine andere Indexzelle gebracht werden.

TXX	i_L	i_R	Transport aus Indexzelle nach Indexzelle	$\langle i_L \rangle := \langle i_R \rangle$ $\langle B \rangle := \langle i_R \rangle$
-----	-------	-------	---	--

Die Größe, die transportiert wurde, steht gleichzeitig im Register B. Dieser Befehl hat die gleiche Wirkung wie die Befehle XB i_R und XC i_L zusammen.

Mit dem Befehl TTX können die Inhalte zweier Indexzellen gegeneinander ausgetauscht werden.

TTX	i_L	i_R	Tausch - Transport in Indexzelle	$\langle B \rangle := \langle i_L \rangle$ $\langle i_R \rangle := \langle i_L \rangle$
-----	-------	-------	-------------------------------------	--

Der Inhalt der links stehenden Indexzelle wird gleichzeitig ins Register B gebracht.

4. ARITHMETISCHE INDEX - OPERATIONEN

Addition und Subtraktion von Indexgrößen werden stets im Register B durchgeführt. Bei einem Teil der Befehle wird das Ergebnis anschließend in eine Indexzelle gebracht. Die nachfolgende Beschreibung der arithmetischen Indexbefehle ist gegliedert nach den Operanden, die miteinander verknüpft werden.

In begrenztem Sinn ist auch eine Multiplikation möglich. Dies ist im letzten Kapitel behandelt.

4.1. Register B und Adressenteil

Bei den Befehlen HBA und VBA wird der Inhalt des Registers B mit dem Adressenteil des Befehls verknüpft. Da Befehle schreibgeschützt sind, wird hier eine Konstante addiert bzw. subtrahiert.

HBA	z	Erhöhe Register B um Adressenteil	$\langle B \rangle := \langle B \rangle + z$
VBA	z	Vermindere Register B um Adressenteil	$\langle B \rangle := \langle B \rangle - z$

$z = 0, \dots, 65\,535$

Der Adressenteil z kann die Werte 0 bis 65 535 annehmen (16 Bits). Eine Modifizierung 2. Art ist nicht möglich.

Um einen größeren Betrag mit dem Register B zu verknüpfen, kann der Befehl auf 1. Art modifiziert oder die Befehle HBC bzw. VBC (siehe Abschnitt 4.2.) können verwendet werden.

```
HBA 3400,
VBA EMIL,
HBA EMIL+5,
```

Bild 4.1 Beispiele zu den Befehlen HBA und VBA

4.2. Register B und Speicherzelle

Mit den Befehlen HBC und VBC werden der Inhalt des Registers B und der Inhalt einer Speicherzelle miteinander verknüpft.

HBC	m	Erhöhe Register B um Speicher	$\langle B \rangle := \langle B \rangle + \langle m \rangle$
VBC	m	Vermindere Register B um Speicher	$\langle B \rangle := \langle B \rangle - \langle m \rangle$

Mit m wird ein Halbwort im Speicher adressiert. Er kann also eine 24-Bit-Größe enthalten.

Mit diesen Befehlen kann auch eine Größe addiert bzw. subtrahiert werden, wenn sie größer als $2^{16}-1$ ist. Die Größe wird über die Angabe einer Konstanten in ein Halbwort im Speicher abgelegt. Die Niederschrift kann in Form eines Literals erfolgen.

```
HBC EMIL,
VBC BERTA+5,
KARL = 361561/H,
HBC KARL,
HBC (361561/H),
```

Bild 4.2 Beispiele zu den Befehlen HBC und VBC

4.3. Register B und Indexzelle

Mit dem Befehl HBPX kann der Inhalt einer Indexzelle zum Register addiert, aber auch vom Register subtrahiert werden. Des Weiteren kann der Inhalt der Indexzelle vorher mit einer Konstanten multipliziert werden.

HBPX	p	i	Erhöhe Register B um p mal Indexzelle	$\langle B \rangle := \langle B \rangle + p \cdot \langle i \rangle$
------	---	---	---------------------------------------	--

$p = \pm 1, \dots, \pm 15$

Hat der Parameter p den Wert +1, so wird der Inhalt der Indexzelle zum Register B addiert, hat er den Wert -1, so wird er vom Register B subtrahiert.

Hat der Parameter p einen größeren Betrag als 1, so wird der Inhalt der Indexzelle mit p multipliziert und anschließend zum Inhalt des Registers B addiert (bei positivem p) bzw. von ihm subtrahiert (bei negativem p).

```
HBPX 1 X1,
HBPX -1 X1,
HBPX 5 X1,
XBA 0,
HBPX 10 X1,
```

Bild 4.3 Beispiele zum Befehl HBPX

Wird das Register B vorher auf 0 gesetzt, so ist die Multiplikation des Inhaltes einer Speicherzelle mit einer Konstanten möglich.

Eine weitere Möglichkeit, den Inhalt der Indexzelle zum Inhalt des Registers B zu addieren, bietet der Befehl RX (siehe Abschnitt 4.4. und 4.7.) in der Kombination RX B i. Er benötigt jedoch längere Zeit als der Befehl HBPX.

M

TAS - Handbuch

Dez. 68

4.4. Register und Indexzelle

Mit dem Befehl RX kann der Inhalt eines der Rechenwerksregister und auch der Inhalt des Registers B mit dem Inhalt einer Indexzelle addiert oder subtrahiert werden.

Es wird jeweils das rechte Halbwort der Rechenwerksregister verknüpft.

RX	s ₁ i	Register u. Indexzelle	$\langle B \rangle := \langle i \rangle \pm \langle s_1 \rangle$ falls s ₃ =C $\langle i \rangle := \langle i \rangle \pm \langle s_1 \rangle$
----	------------------	------------------------	---

s₁: A, Q, D, H oder B
s₂: leer = positiv + statt +
N = negativ -
s₃: leer = nicht zurückspeichern
C = zurückspeichern

Wird zusätzlich die Spezifikation C angegeben, so wird das Ergebnis in die Indexzelle zurückgespeichert, d.h. also, daß die Indexzelle um den Inhalt des in der Spezifikation angegebenen Registers erhöht bzw. vermindert wird.

RX	AN	X1,
RX	NH	X1,
RX	BC	X1,

Bild 4.4 Beispiele zum Befehl RX

4.5. Indexzelle und Parameter

Beim Befehl HXP wird zum Inhalt einer Indexzelle der im Adressenteil der Befehle stehende Wert des Parameters p hinzuaddiert (wenn p positiv) bzw. davon subtrahiert (wenn p negativ).

HXP	p i	Erhöhe Indexzelle um Parameter	$\langle B \rangle := \langle i \rangle + p$ $\langle i \rangle := \langle i \rangle + p$
-----	-----	--------------------------------	--

p : 0...+127

Das Ergebnis steht gleichzeitig im Register B.

HXP	5	X1,
HXP	-10	X1,

Bild 4.5 Beispiele zum Befehl HXP

4.6. Indexzelle und Indexzelle

Mit den Befehlen HXX und VXX werden die Inhalte zweier Indexzellen miteinander verknüpft.

HXX	i _L i _R	Erhöhe Indexzelle um Indexzelle	$\langle B \rangle := \langle i_R \rangle + \langle i_L \rangle$ $\langle i_R \rangle := \langle i_R \rangle + \langle i_L \rangle$
VXX	i _L i _R	Vermindere Indexzelle um Indexzelle	$\langle B \rangle := \langle i_R \rangle - \langle i_L \rangle$ $\langle i_R \rangle := \langle i_R \rangle - \langle i_L \rangle$

Beim Befehl HXX wird der Inhalt der links angegebenen Indexzelle zum Inhalt der rechts angegebenen Indexzelle addiert, und beim Befehl VXX wird der Inhalt der links angegebenen Indexgröße von der rechts angegebenen subtrahiert. In beiden Fällen steht das Ergebnis in der rechts angegebenen Indexzelle und auch im Register B.

HXX	X2	X1,
VXX	X2	X1,

Bild 4.6 Beispiele zu den Befehlen HXX und VXX

4.7. Multiplizieren

Sollte eine Multiplikation bei den Indexgrößen notwendig werden, so ist dies nur begrenzt oder über Umwegen möglich.

Eine Multiplikation mit einer Konstanten, deren Betrag nicht größer ist als 15, kann mit dem Befehl HBPX (siehe Abschnitt 4.3.) durchgeführt werden. Durch Modifikation 1. Art läßt sich die Konstante auch verändern.

Eine Multiplikation mit dem Faktor 2 bzw. -2 ist mit dem Befehl RX i bzw. RX BN i möglich.

Eine Multiplikation mit einer 2er-Potenz kann durch Schiften mit dem Befehl SHB (siehe Abschnitt 6.) vorgenommen werden. Eine Division ist ebenfalls mit diesem Befehl möglich.

In allen anderen Fällen ist es nötig, die Operanden mit Hilfe des Befehls R im Rechenwerk miteinander zu verknüpfen und mit dem Befehl RX in das Befehlswerk zurückzuholen.

5. ABFRAGEN - VERZWEIGUNGEN - SPRÜNGE

Zur Verzweigung von Programmen und zur Bildung von Schleifen stehen die Sprungbefehle zur Verfügung. Eine Beschreibung aller Sprungbefehle ist unter dem Abschnitt Sonstige Operationen zu finden. Der Aufbau von Schleifen ist im Abschnitt 11. beschrieben.

Eine Gruppe von Sprungbefehlen vergleicht das Register B bzw. eine Indexzelle mit dem Wert Null. Es sind dies die in Bild 5.1 angegebenen Befehle.

```
SXN ANF,
SXGG -2R,
SXX 15R,
```

Bild 5.2 Beispiele für Sprungbefehle

Ist die Sprungbedingung erfüllt, so wird ein Sprung auf die im Adressenteil des Befehls angegebene Adresse ausgeführt. Im anderen Fall hat der Befehl keine Wirkung; es wird also im Programm fortgefahren.

Der Befehl SZX ist besonders geeignet zur Bildung von Schleifen mittels einer Zählgröße. Er vergleicht, ob die Zählgröße bereits 0 ist. Ist das nicht der Fall, so zählt er um 1 herauf, und die Schleife wird erneut durchlaufen (siehe Abschnitt 11.1.).

```

      :
      : ZX -10 X1,
A =   M X1,
      B ANF,
      :
      :
S =   SZX A-S X1,
      :
      :
```

Bild 5.3 Beispiel zum Befehl SZX

SXI	m	Springe, wenn Index <u>i</u> identisch 0	$\langle B \rangle = \pm 0$
SXN	m	<u>n</u> icht identisch 0	$\langle B \rangle \neq \pm 0$
SXG	m	<u>g</u> rößer 0	$\langle B \rangle > \pm 0$
SXGG	m	<u>g</u> leich oder <u>g</u> rößer 0	$\langle B \rangle \geq \pm 0$
SXX	m	<u>k</u> leiner 0	$\langle B \rangle < \pm 0$
SXKG	m	<u>k</u> leiner oder <u>g</u> leich 0	$\langle B \rangle \leq \pm 0$
SXR	m	Springe, wenn Index <u>r</u> echtes Bit L	$\langle B \rangle_{24} = L$
SXRN	m	<u>r</u> echtes Bit <u>n</u> icht L	$\langle B \rangle_{24} = 0$
SZX	p i	Springe und <u>z</u> ähle, wenn Index <u>k</u> leiner 0	$\langle i \rangle < \pm 0$

p: $\pm 0 \dots \pm 127$

Bild 5.1 Sprungbefehle zur Abfrage von Indexgrößen

6. SCHIFTEN

Im Register B kann mit dem Befehl SHB geschifft werden.

SHB	s p	Schifte im Register B	$\langle B \rangle := \langle B \rangle$ geschifft um p Stellen
-----	-----	-----------------------	---

s:Schiftrichtung
 R = rechts
 L = links
 p:Anzahl der Binärstellen 0...255

Im Adressenteil des Befehls ist angegeben, ob nach rechts oder links geschifft werden soll und um wieviel Binärstellen. Die herausgeschifteten Stellen gehen verloren; auf der anderen Seite des Registers werden Null-Bits nachgezogen.

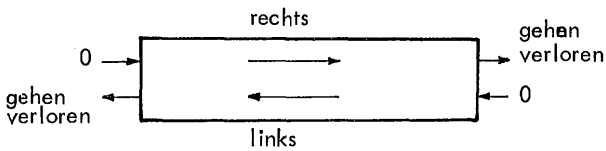


Bild 6.1 Schiften im Register B

Geschifft werden können nur positive Wörter, da stets Null-Bits nachgezogen werden. Wird z.B. eine negative Null um 1 Stelle nach rechts geschifft, so ergibt sich folgendes Bild:

vor dem Schift $\llcorner LL\dots LL \triangleq -0$
 1 Stelle nach rechts geschifft $\circ LL\dots LL \triangleq +(2^{23} - 1)$

Ein Schift um p Stellen nach links bedeutet Multiplikation mit 2^p und nach rechts Division durch 2^p .

SHB	R	12,	} ungerade Zahl wird um 1 erniedrigt
SHB	R	1,	
SHB	L	1,	} ergibt Anfangsadresse der Großseite
SHB	R	16,	
SHB	L	16,	

Bild 6.2 Beispiele zum Befehl SHB



7. MODIFIZIEREN

Mit Hilfe eines Modifizierbefehls kann der Adressenteil eines nachfolgenden Befehls verändert werden, indem zum Adressenteil eine Größe hinzuaddiert wird.

Durch die Modifizierung wird der Adressenteil des nachfolgenden Befehls während der Abrufphase verändert und zwar im Befehlswerk. Im Speicher bleibt der Befehl jedoch unverändert, das heißt er bleibt so stehen, wie er im Programm niedergeschrieben wurde.

Mit Hilfe der Modifizierbefehle und auch der im Abschnitt 9 und 10 beschriebenen Ersetzbefehle ist es möglich, die beim Programmieren notwendigen Schleifen zu organisieren. Bei jedem Durchlauf der Schleife wird eine Größe herauf- oder heruntergezählt. Diese Größe kann mit Hilfe eines Modifizierbefehls auf den Adressenteil des nachfolgenden Befehls addiert werden, so daß sich dessen Adresse laufend erhöht oder erniedrigt, und so auf eine Folge von Operanden zuge-

griffen werden kann. Bei allen Befehlen, solange sie im Speicher sind, hat der Adressenteil eine Länge von 16 Bits. Werden sie ins Befehlswerk geholt, so wird der Adressenteil auf eine Länge von 24 Bits erweitert, indem links mit Nullen erweitert wird. Mit Hilfe der Modifizier- und Ersetzbefehle kann nun die volle Länge des Adressenteils von 24 Bits belegt werden und damit der gesamte Kernspeicher adressiert werden,

Die von einem Modifizierbefehl erzeugte Größe, die den nachfolgenden Befehl modifizieren soll, wird Modifikator genannt. Es werden zwei Arten von Modifikatoren unterschieden:

- Modifikator 1. Art = mod1
- Modifikator 2. Art = mod2

und entsprechend gibt es Befehle, die einen Modifikator 1. Art, und Befehle, die einen Modifikator 2. Art

vorhergehender Befehl
(Adresse n-1)

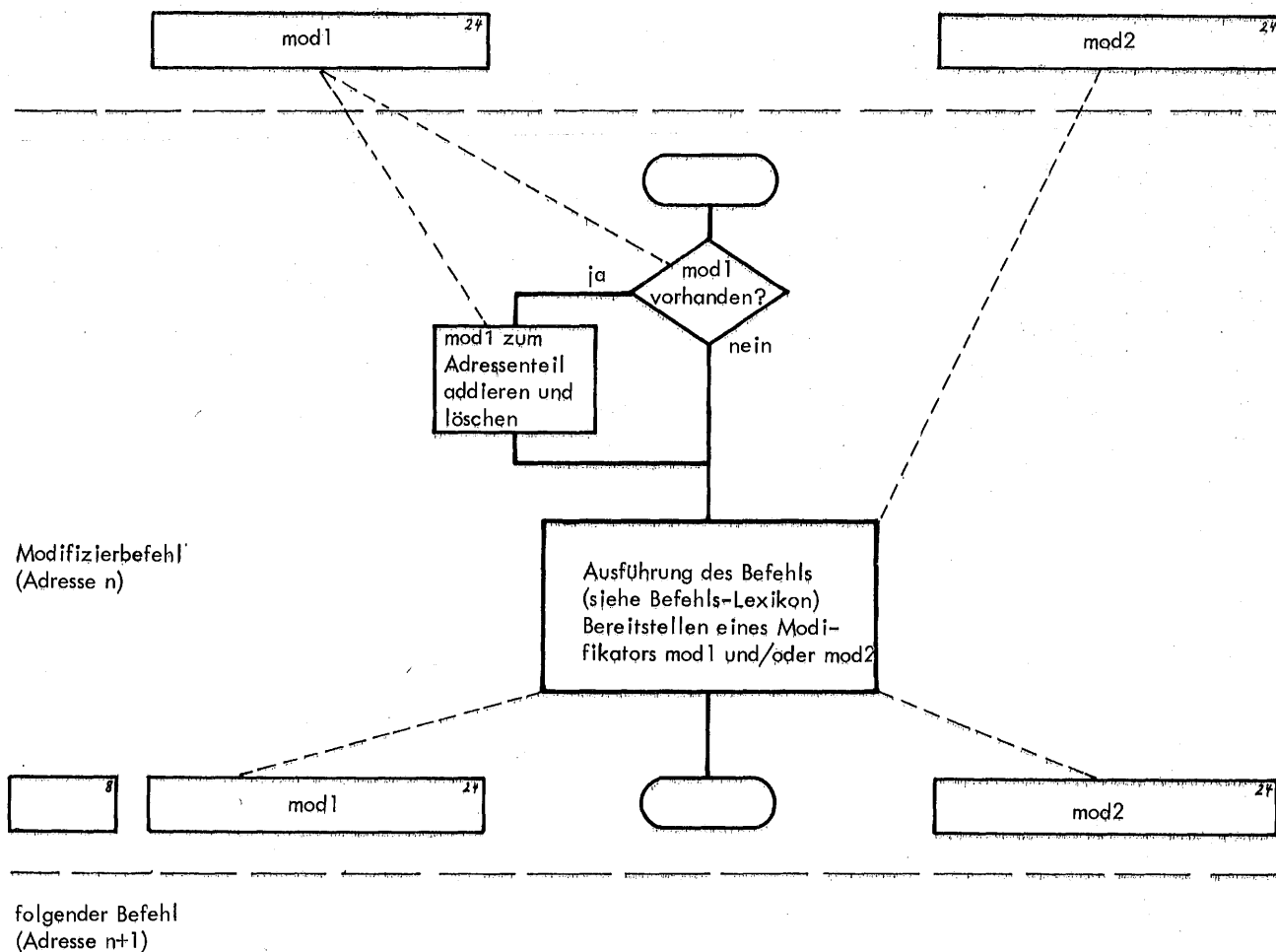
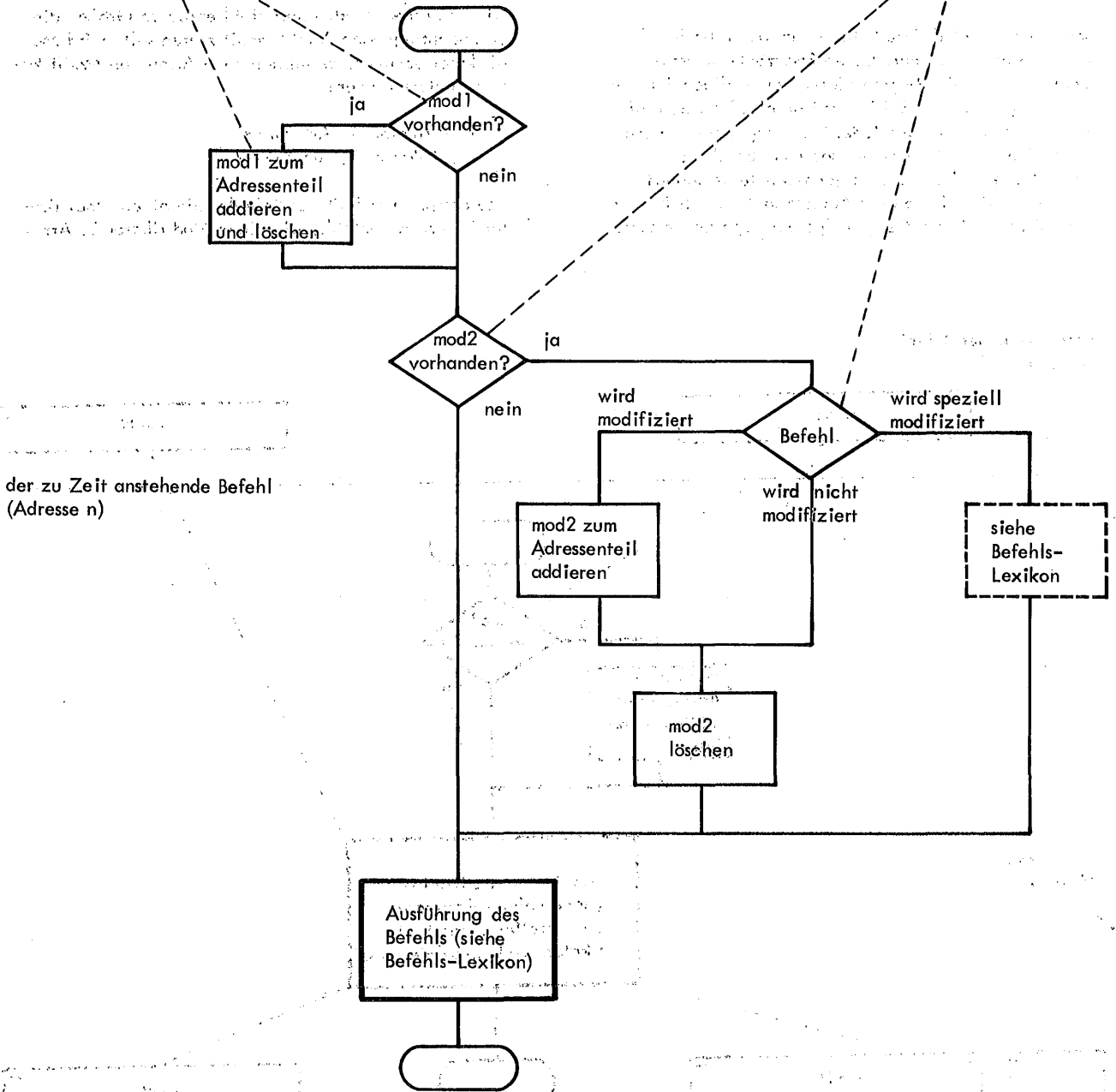


Bild 7.1 Wirkung eines Modifizierbefehls

vorhergehender Befehl
(Adresse n-1)

mod1

mod2



der zu Zeit anstehende Befehl
(Adresse n)

folgender Befehl
(Adresse n+1)

Bild 7.2 Modifizieren eines Befehls

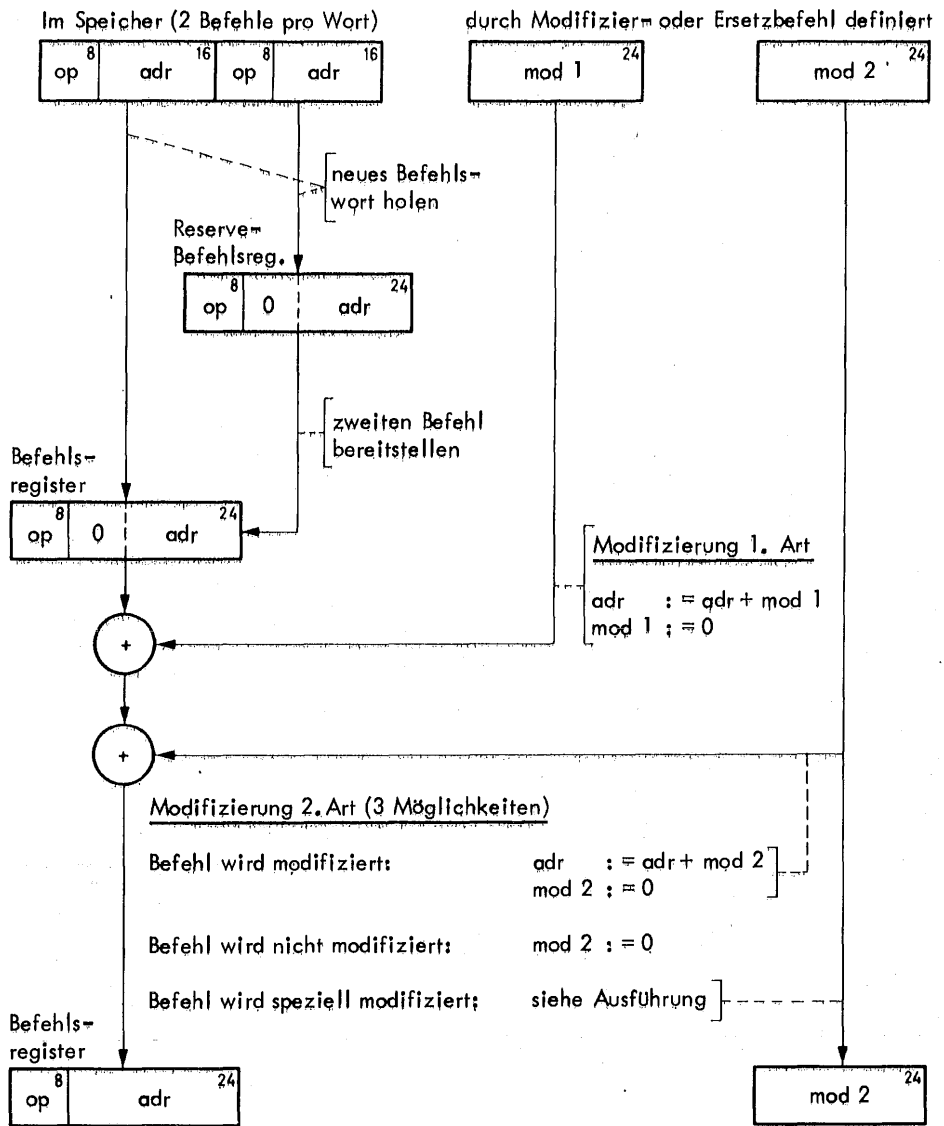


Bild 7.3 Adressenrechnung in der Abrufphase

erzeugen. Der Befehl MD liefert beide Modifikatoren gleichzeitig. Alle Befehle werden in Bezug auf den Modifikator 2. Art in drei Gruppen unterteilt:

- Befehle, die auf 2. Art modifiziert werden
- Befehle, die auf 2. Art nicht modifiziert werden
- Befehle, die auf 2. Art speziell modifiziert werden

Bei der ersten Gruppe besteht kein Unterschied zwischen der Modifizierung 1. und 2. Art. Die zweite Gruppe kann auf 2. Art nicht modifiziert werden; ein vorhergehender Modifizierbefehl 2. Art ist also bedeutungslos. Bei der dritten Gruppe tritt der Unterschied der beiden Arten deutlich hervor. Es sind dies ein Teil der Ersetzbefehle und ein Teil der Modifizierbefehle selbst. Bei diesen Befehlen wirkt ein Modifikator 1. Art immer auf den Adressenteil des Befehls, während ein Modifikator 2. Art bei den Modifizierbefehlen zum jeweiligen Modifikator hinzugezählt wird und bei den Ersetzbefehlen auf den Zweitbefehl wirkt.

7.1, Modifizierung 1. Art

Ein Modifikator 1. Art (mod1) wird immer auf den Adressenteil (adr) des nachfolgenden Befehls addiert und anschließend gelöscht. Es kann also mit einem Modifizierbefehl 1. Art jeder Befehl modifiziert werden. Als Befehle stehen

MF, MFU
MCF, MCFU

zur Verfügung, außerdem der Befehl MD, der einen Modifikator 1. und 2. Art erzeugt.

7.2, Modifizierung 2. Art

Für die Modifizierung 2. Art stehen die Befehle

M, MH, MHX, MRX
MC, MCE,
MA, MNA



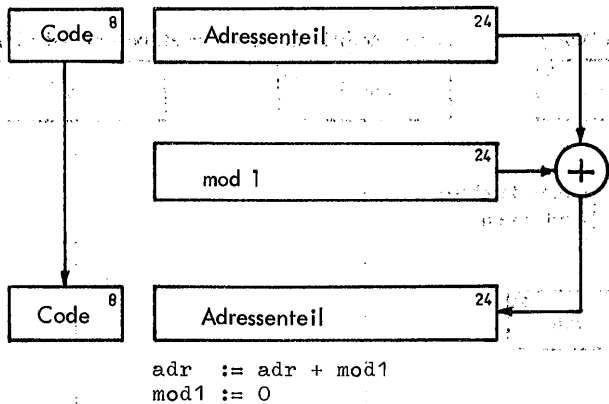


Bild 7.4 Addieren eines Modifikators 1. Art

zur Verfügung. Sie erzeugen einen Modifikator 2. Art (mod2). Wie der Modifikator 2. Art auf den nachfolgenden Befehl wirkt, hängt von diesem nachfolgenden Befehl ab. Dabei werden drei Befehlsgruppen unterschieden:

- Befehle, die auf 2. Art modifiziert werden
- Befehle, die auf 2. Art nicht modifiziert werden
- Befehle, die auf 2. Art speziell modifiziert werden

In den folgenden 3 Abschnitten wird die Wirkung eines Modifikators 2. Art auf diese Befehlsgruppen näher erläutert.

Bei der Beschreibung der Wirkung der Befehle im "Befehls-Lexikon" ist bei jedem Befehl unter "bei mod2" angegeben, wie er durch mod2 modifiziert wird.

In der großen Befehlsliste ist in der Spalte "mod2" im ersten Fall ein Pluszeichen angegeben und im letzten Falle ein "sp". Wird der Befehl nicht modifiziert, so ist die Spalte freigelassen.

Befehls-Lexikon: "bei mod2"	Große Befehlsliste: "mod2"
wird modifiziert	+
wird nicht modifiziert	
wird speziell modifiziert	sp

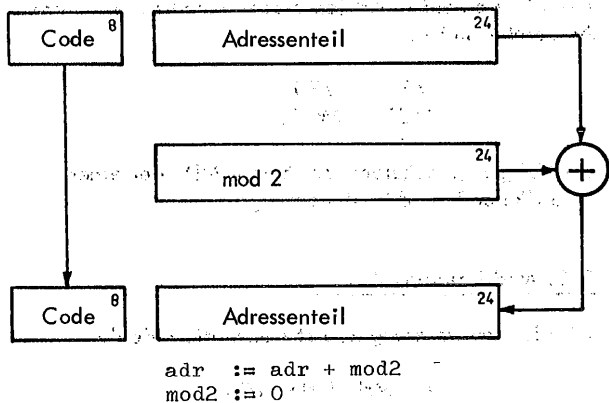


Bild 7.5 Addieren eines Modifikators 2. Art

7.2.1. Befehl wird modifiziert

Bei diesen Befehlen wirkt der Modifikator 2. Art ebenso wie der Modifikator 1. Art, d.h. er wird in der Abrufphase des Befehls zum Adressenteil hinzuaddiert. Bei diesen Befehlen ist es also gleich, ob auf 1. oder 2. Art modifiziert wird

7.2.2. Befehl wird nicht modifiziert

Bei diesen Befehlen wird ein Modifikator 2. Art nicht addiert. Der Modifikator 2. Art wird gelöscht. Die Wirkung ist die gleiche, als ob der Befehl nicht auf 2. Art modifiziert wäre.

Diese Befehle können also nur auf 1. Art modifiziert werden.

7.2.3. Befehl wird speziell modifiziert

Zu diesen Befehlen gehören

- ein Teil der Modifizierbefehle
- ein Teil der Ersetzbefehle
- die Befehle BNZ, CNZ, SE, SUE, US

Bei diesen Befehlen wird der Modifikator nicht direkt addiert sondern beeinflusst auf ganz bestimmte Weise den Befehl, und zwar nicht in der Abrufphase, sondern in der Ausführungsphase des Befehls.

Welche Wirkung der Modifikator 2. Art hat, ist aus der Beschreibung des Befehls im Befehls-Lexikon bzw. in der Großen Befehlsliste zu entnehmen.

Bei den speziell modifizierten Befehlen kommt der Unterschied zwischen der Modifizierung 1. Art und 2. Art voll zum tragen. Im Bild 7.6. ist ein Beispiel gezeigt, in dem der Befehl E auf beide Arten modifiziert wird. Bei 1. Art wird erst die Adresse X2 um den Inhalt der

Ausgangswerte: $\langle X1 \rangle = 2$
 $\langle X2 \rangle = 1000$
 $\langle X2+2 \rangle = 2000$

1. Art:

```

:
:
MF X1,
E B X2,
:
:
mod1:
  <X1> = 2
Zweitbefehl:
B <X2+<X1>> B 2000

```

2. Art:

```

:
:
M X1,
E B X2,
:
:
mod2:
  <X1> = 2
Zweitbefehl:
B <X2>+<X1> B 1002

```

Bild 7.6 Beispiel Modifizierung von Befehl E auf 1. und 2. Art

Indezzelle X1 erhöht, und somit hat der Ersetzbefehl die Form $E B X2 + X1$. Im Beispiel wird der Adressenteil des Zweitcodes aus der Indexzelle $X2 + 2$ geholt; das ist die Zahl 2000. Bei 2. Art wird der Adressenteil des Zweitcodes aus der Indexzelle X2 geholt und dazu der mod2, d.i. der Inhalt des Speicherzelle X1 hinzuaddiert. Im Beispiel ist also der Adressenteil des Zweitcodes 1002.

Ausgangswerte:

$\langle X1 \rangle$	=	2
$\langle 25 \rangle$	=	1000
$\langle 27 \rangle$	=	2000

1. Art:

<pre> : : MF X1, MC 25, : : </pre>	<p>mod1:</p> <table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px 10px;">$\langle X1 \rangle$</td> <td style="padding: 0 5px;">=</td> <td style="border: 1px solid black; padding: 2px 10px; text-align: center;">2</td> </tr> </table> <p>mod2:</p> <table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px 10px;">$\langle 25 + \langle X1 \rangle \rangle$</td> <td style="padding: 0 5px;">=</td> <td style="border: 1px solid black; padding: 2px 10px; text-align: center;">2000</td> </tr> </table>	$\langle X1 \rangle$	=	2	$\langle 25 + \langle X1 \rangle \rangle$	=	2000
$\langle X1 \rangle$	=	2					
$\langle 25 + \langle X1 \rangle \rangle$	=	2000					

2. Art:

<pre> : : M X1, MC 25, : : </pre>	<p>mod2:</p> <table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px 10px;">$\langle X1 \rangle$</td> <td style="padding: 0 5px;">=</td> <td style="border: 1px solid black; padding: 2px 10px; text-align: center;">2</td> </tr> </table> <p>mod2:</p> <table style="border-collapse: collapse;"> <tr> <td style="border: 1px solid black; padding: 2px 10px;">$\langle 25 \rangle + \langle X1 \rangle$</td> <td style="padding: 0 5px;">=</td> <td style="border: 1px solid black; padding: 2px 10px; text-align: center;">1002</td> </tr> </table>	$\langle X1 \rangle$	=	2	$\langle 25 \rangle + \langle X1 \rangle$	=	1002
$\langle X1 \rangle$	=	2					
$\langle 25 \rangle + \langle X1 \rangle$	=	1002					

Bild 7.7 Beispiel Modifizierung von Befehl MC auf 1. und 2. Art

Bei der Modifizierung 1. Art wird der Inhalt der Indexzelle X1 zur Adresse 25 des Befehls MC addiert und dann aus der Speicherzelle 27 der Modifikator 2. Art geholt. Bei der Modifizierung 2. Art wird der Inhalt der Speicherzelle 25 als Modifikator 2. Art geholt und der Modifikator 2. Art vom Vorbefehl (Inhalt der Indexzelle $X1 = 2$) hinzuaddiert.



8. MODIFIZIERBEFEHLE

Die einzelnen Modifizierbefehle unterscheiden sich in der Art, wie sie den Modifikator erzeugen:

- Modifikator aus Indexzelle
- Modifikator aus dem Adressenteil
- Modifikator aus dem Speicher

In den nachfolgenden Abschnitten sind die Modifizierbefehle in dieser Gruppierung beschrieben, dabei sind die Modifizierbefehle 1. und 2. Art, die den Modifikator auf gleiche Art erstellen, gemeinsam behandelt.

8.1. Modifikator aus Indexzelle

Die nachstehenden Befehle holen den Modifikator aus einer Indexzelle.

M	i	Modifiziere	$\langle B \rangle := \langle i \rangle + \text{mod2}$ $\text{mod2} := \langle i \rangle + \text{mod2}$
MF	i	Modifiziere in jedem Fall	$\text{mod1} := \langle i \rangle + \text{mod2}$ $\langle B \rangle := \langle i \rangle + \text{mod2}$
MFU	i	Modifiziere in jedem Fall mit unverändertem B	$\text{mod1} := \langle i \rangle + \text{mod2}$

Der Befehl M erzeugt einen Modifikator 2. Art, die anderen beiden Befehle einen Modifikator 1. Art.

Der Befehl MFU verändert das Register B nicht, während bei den beiden anderen Befehlen der Modifikator gleichzeitig im Register B steht.

Ein vom Vorbefehl vorhandener Modifikator 2. Art wird zum Modifikator hinzuaddiert. Dies bedeutet bei der Befehlsfolge

```
M X1,
M X2,
M X3,
```

daß der letzte Modifikator die Summe der Inhalte der Indexzellen X1 bis X3 ist.

Bei dem Befehl MH wird zusätzlich noch eine Zahl p zum Modifikator addiert, und auch der Inhalt der Indexzelle wird um die Zahl p erhöht. Der Modifikator steht gleichzeitig im Register B.

MH	p i	Modifiziere nach Erhöhung	$\langle B \rangle := \langle i \rangle + p$ $\langle i \rangle := \langle i \rangle + p$ $\text{mod2} := \langle i \rangle + p$
----	-----	---------------------------	--

p = Parameter 0 ... + 127

Dieser Befehl eignet sich dazu, innerhalb einer Schleife einen Befehl zu modifizieren, und gleichzeitig eine laufende Erhöhung der Modifiziergröße zu erreichen. Der Befehl kann nicht auf 2. Art modifiziert werden.

Im Beispiel Bild 8.1 ist ein Block von 100 Ganzwörtern angenommen. Jedes 5. Wort soll zum Inhalt des Registers A addiert werden. Es sind dies die Wörter mit den Adressen SUM, SUM + 10, . . . , SUM + 190. Die Indexzelle X1 wird auf den Wert 200 gesetzt und durch den Befehl MH am Anfang der Schleife um 10 vermindert. Der verminderte Wert wird auf den Adressenteil des Befehls A addiert. So wird beim ersten Durchlauf der Inhalt der Zelle SUM + 190 zum Inhalt des Registers A addiert. Das Programm beginnt also am Ende des Blockes mit der Addition.

```

XBA 200,
XC X1,
LR A1,
ANF= MH -10 X1,
      A SUM,
      SXN ANF,

```

Durchlauf	Inhalt X1	Adressenteil von Befehl A
1.	200 - 10 = 190	SUM + 190
2.	190 - 10 = 180	SUM + 180
3.	180 - 10 = 170	SUM + 170
usw.		
19.	20 - 10 = 10	SUM + 10
20.	10 - 10 = 0	SUM

Bild 8,1 Beispiel zum Befehl MH

Im Beispiel Bild 8.2 wird das jeweils 5. Ganzwort der Blöcke mit der Anfangsadresse SUM1 und SUM2 addiert und in das 5. Ganzwort des Blockes mit der Anfangsadresse SUM gebracht. Der erste Befehl (Befehl B) wird mit MH modifiziert, und dabei wird die Indexgröße hochgezählt. Die weiteren Befehle (Befehle A und C) brauchen nur noch mit M modifiziert zu werden.

Während im Beispiel Bild 8.1 am Ende des Blockes begonnen wurde, wird im Beispiel 8.2 am Anfang der Blöcke begonnen. Dies wurde dadurch erreicht, daß die Indexzelle auf den negativen Wert gesetzt und mit MH hochgezählt wurde. Um dies auszugleichen, muß in den Adressenteilen der Befehle B, A und C die Adresse des letzten Wortes angegeben werden.



TAS - Handbuch

Dez. 68

```

:
:
XBAN 200,
XC X1,
ANF= MH 10 X1,
      B SUM1+190,
      M X1,
      A SUM2+190,
      M X1,
      C SUM+190,
      SXN ANF,
:
:

```

$\langle \text{SUM1+a} \rangle + \langle \text{SUM2+a} \rangle = \langle \text{SUM+a} \rangle$
 $a = 0, 10, 20, \dots, 180, 190.$

Durchlauf	Inhalt X1	Adressenteil von Befehl C
1.	-200 + 10 = -190	SUM
2.	-190 + 10 = -180	SUM + 10
3.	-180 + 10 = -170	SUM + 20
usw.		
19.	-20 + 10 = -10	SUM + 180
20.	-10 + 10 = 0	SUM + 190

Bild 8.2 Beispiel zu den Befehlen M und MH

Eine Erweiterung des Befehls MH stellt der Befehl MHX dar. Bei ihm wird statt um den Wert p um den Inhalt einer Indezzelle erhöht.

MHX	$i_L i_R$	Modifiziere nach Erhöhung um Indezzelle	$\text{mod2} := \langle i_L \rangle + \langle i_R \rangle$ $\langle B \rangle := \langle i_L \rangle + \langle i_R \rangle$ $\langle i_R \rangle := \langle i_L \rangle + \langle i_R \rangle$
-----	-----------	---	--

Damit ist es möglich, gegenüber dem Befehl MH die Schrittweite im Laufe des Programmes zu ändern, indem die Indezzelle i_L auf einen anderen Wert gesetzt wird.

Eine Modifizierung auf 2. Art ist nicht möglich.

```

:
:
XBA 200,
XC X1,
ZX -10 X2,
LR A1,
MHX X2 X1,
A SUM,
SXN -2R,
XBA 200,
XC X1,
ZX -20 X2,
LR A1,
MHX X2 X1,
A SUM,
SXN -2R,
:
:

```

Bild 8.3 Beispiel zum Befehl MHX

Das Beispiel im Bild 8.1 wurde im Bild 8.3 auf den Befehl MHX abgewandelt. Durch den Befehl ZX -10 X2 wurde eine Schrittweite von 5 Ganzwörtern (10 Halbwörtern) eingestellt und mit ZX -20 X2 eine Schrittweite von 10 Ganzwörtern (20 Halbwörtern).

Mit Hilfe des Befehls MRX ergibt sich der Modifikator 2. Art aus der Summe des Inhalts einer Indezzelle und dem Inhalt eines der Register A, Q, D, H oder B.

MRX	s i	Modifiziere mit Register und Indezzelle	$\text{mod2} := \langle i \rangle \pm \langle s_1 \rangle$ $\langle B \rangle := \langle i \rangle \pm \langle s_1 \rangle$ falls $s_3 = C$: $\langle i \rangle := \langle i \rangle \pm \langle s_1 \rangle$
-----	-----	---	---

s_1 : Register A, Q, D, H oder B
 s_2 : leer: + } statt ±
N : - }
 s_3 : leer: nicht zurückspeichern
C : zurückspeichern

Wird beim Befehl MRX zusätzlich zur Registerangabe ein C hinzugefügt, so wird eine Variante des Befehls MHX erreicht und es kann in der Indezzelle mit dem Inhalt des Registers hochgezählt werden.

Wird zu den Registerangaben ein N hinzugefügt, so wird der Inhalt des Registers nicht addiert, sondern subtrahiert (negativer Wert).

Eine Modifizierung nur mit dem Inhalt eines Registers kann über den Registerbefehl R (siehe Ersetzbefehle) in Verbindung mit dem Befehl MC (siehe Abschnitt 7.2) z. B. durch R MC A oder R MC B erreicht werden.

```

MRX A X1,
MRX HN X2,
MRX B X1,
MRX Q X2,
MRX HCN X3,

```

Bild 8.4 Beispiel für Befehl MRX

8.2. Modifikator aus Adressenteil

Die Befehle MA und MNA bilden den Modifikator aus ihrem eigenen Adressenteil. Da Befehle im allgemeinen schreibgeschützt sind, handelt es sich bei der Zahl z um eine Konstante.

MA	z	Modifiziere mit Adressenteil	$\langle B \rangle := z + \text{mod2}$ $\text{mod2} := z + \text{mod2}$
MNA	z	Modifiziere mit negativem Adressenteil	$\langle B \rangle := -z + \text{mod2}$ $\text{mod2} := -z + \text{mod2}$

z: 0...65535 (vor Modifizierung)

Beim Befehl MA wird der Adressenteil z zum Modifikator, und beim Befehl MNA ist der Modifikator der Adressenteil z mit negativem Vorzeichen. Ein vom vorhergehenden Befehl vorhandener Modifikator wird zum neuen Modifikator addiert.

Diese Befehle eignen sich besonders, um einen Befehl mit einer Konstanten zu modifizieren, wenn diese Konstante eine Länge von 16 Bits hat, d.h. sie muß kleiner als 2^{16} sein, bzw. die symbolische Adresse muß in der Großseite 0 liegen. Ist sie größer, so muß mit den Befehlen aus Abschnitt 8.3 gearbeitet werden.

MA	ANF,
E	B X3,
M	X4,
MA	ANF,
E	B X3,

Bild 8.5 Beispiele zum Befehl MA

Im ersten und zweiten Beispiel Bild 8.5 wird zur Anfangsadresse ANF die Indexgröße aus der Speicherzelle X3 addiert. Im zweiten Fall wird zu der wie im ersten Beispiel errechneten Adresse noch der Inhalt der Indexzelle X4 addiert.

8.3. Modifikator aus Speicher

Die nachstehenden Modifizierbefehle holen den Modifikator aus dem Speicher und zwar aus einem Halbwort, Es ist also auch eine 24-Bit-Größe möglich.

MC	m	Modifiziere aus Speicher	$\langle B \rangle := \langle m \rangle + \text{mod}2$ $\text{mod}2 := \langle m \rangle + \text{mod}2$
MCF	m	Modifiziere aus Speicher in jedem Fall	$\text{mod}1 := \langle m \rangle + \text{mod}2$ $\langle B \rangle := \langle m \rangle + \text{mod}2$
MCFU	m	Modifiziere aus Speicher in jedem Fall mit unverändertem Register B	$\text{mod}1 := \langle m \rangle + \text{mod}2$

Der Befehl MC modifiziert auf 2. Art, die anderen beiden auf 1. Art. Bei den Befehlen MC und MCF ist der Modifikator gleichzeitig im Register B, während beim Befehl MCFU das Register B nicht verändert wird.

Bei allen Befehlen wird ein vom vorhergehenden Befehl vorhandener Modifikator 2. Art hinzuaddiert.

Diese Befehle eignen sich besonders, um einen Befehl mit einer Konstanten, die eine Länge von 24 Bits hat, zu modifizieren. Die Konstante kann dabei als Literal geschrieben werden.

Das Beispiel im Bild 8.6 ist eine Abwandlung des Beispiels im Bild 8.5. Es ist davon ausgegangen, daß die Adresse ANF eine 24 Bit lange Adresse sein kann.

AK =	ANF/A,
MC	AK,
E	B X3,
MC	(ANF/A),
E	B X3,
M	X4,
MC	(ANF/A),
E	B X4,

Bild 8.6 Beispiele zum Befehl MC

Der Befehl MCE ist eine sehr spezielle Variation des Befehls MC. Er bildet den Modifikator aus einer Kette von Ersetzungen.

MCE	m	Modifiziere aus Speicher nach Ersetzung	$\langle B \rangle := \langle \dots \langle m \rangle \dots \rangle + \text{mod}2$ Abbruch wenn: $\langle \dots \langle m \rangle \dots \rangle_1 = L$ $\langle B \rangle := \langle B \rangle_2$ $\text{mod}2 := \langle B \rangle$
-----	---	---	--

Es wird der Inhalt der Speicherzelle m wiederum als Adresse eines Halbwortes aufgefaßt. Der Inhalt dieses Halbwortes ergibt die Adresse eines weiteren Halbwortes, das wiederum als Adresse aufgefaßt wird usw. Diese Ersetzungskette wird abgebrochen, wenn das erste Bit eines Halbwortes gesetzt ist.

Der Inhalt dieses Halbwortes wird der Modifikator. Dabei wird das 1. Bit dem 2. angeglichen. Ist vom vorhergehenden Befehl ein Modifikator 2. Art vorhanden, so wird er addiert. Der neue Modifikator steht gleichzeitig im Register B.

Die Wirkungsweise wird anhand eines Flußdiagramms im Bild 8.7 erläutert.

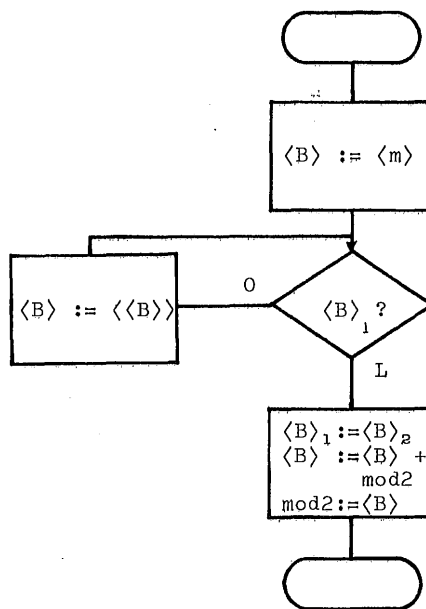


Bild 8.7 Ablauf des Befehls MCE



Ausgangswerte: $\langle 15 \rangle = 23$
 $\langle 23 \rangle = 13$
 $\langle 13 \rangle = 64$
 $\langle 64 \rangle = 333 + 1. \text{ Bit gesetzt}$
 $\cong 333 + 8 \cdot 388 \cdot 608 \text{ bzw.}$
 $\cong 8 \cdot 388 \cdot 274$

MCE 15,

Ergebnis: $\langle B \rangle := 333$
mod2 := 333

Bild 8.8 Beispiel zum Befehl MCE

9. ERSETZEN

Alle Befehle werden im Normalfall in einen schreibgeschützten Teil des Speichers gelegt; damit sind sie nicht veränderlich. Es besteht jedoch häufig die Notwendigkeit, daß der Adressenteil eines Befehls vom Programm selbst laufend verändert werden muß. Diese Möglichkeiten bieten die Ersetzbefehle.

Neben ihrem Code haben die Ersetzbefehle in der linken Hälfte ihres Adressenteils einen Zweitcode, der jeder

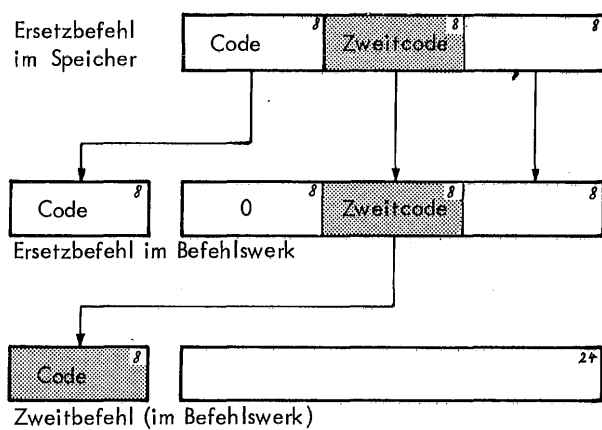


Bild 9.1 Prinzip der Doppelcodebefehle (Ersetzbefehle)

Code des Rechners sein kann. Sie werden daher auch Doppelcodebefehle genannt.

Die Ersetzbefehle bewirken, daß ein zweiter Befehl erzeugt wird – der Zweitbefehl – der unmittelbar nach dem Ersetzbefehl ausgeführt wird. Als Code dieses Zweitbefehls gilt der Zweitcode.

Der Adressenteil des Zweitcodes kann auf verschiedene Weise erzeugt werden und ist abhängig vom Code des Ersetzbefehls.

Im Ablauf des Programms wirkt sich das so aus, als ob der Ersetzbefehl durch den von ihm erzeugten Befehl ersetzt wird, also an der gleichen Stelle im Programm steht. Dies ist von Bedeutung, wenn der Zweitbefehl ein relativer Sprung ist.

Für den Zweitbefehl braucht die Abrufphase nicht voll durchlaufen zu werden. Von den in der "Großen Befehlsliste", Seite 2, angegebenen Zeiten entfällt auf jeden Fall die Befehlsabrufzeit von 8 Takten und die Zeit für die Modifizierung 1. Art von 8 Takten. Bei den Befehlen EZ und ENZ liegt die Zeit für eine Modifizierung auf 2. Art in der Ausführungszeit des Befehls. Die Befehle E, EMU, RLR und T geben den Modifikator 2. Art (mod2) an den Zweitbefehl weiter; sie brauchen für die Modifizierung 2. Art keine Zeit.

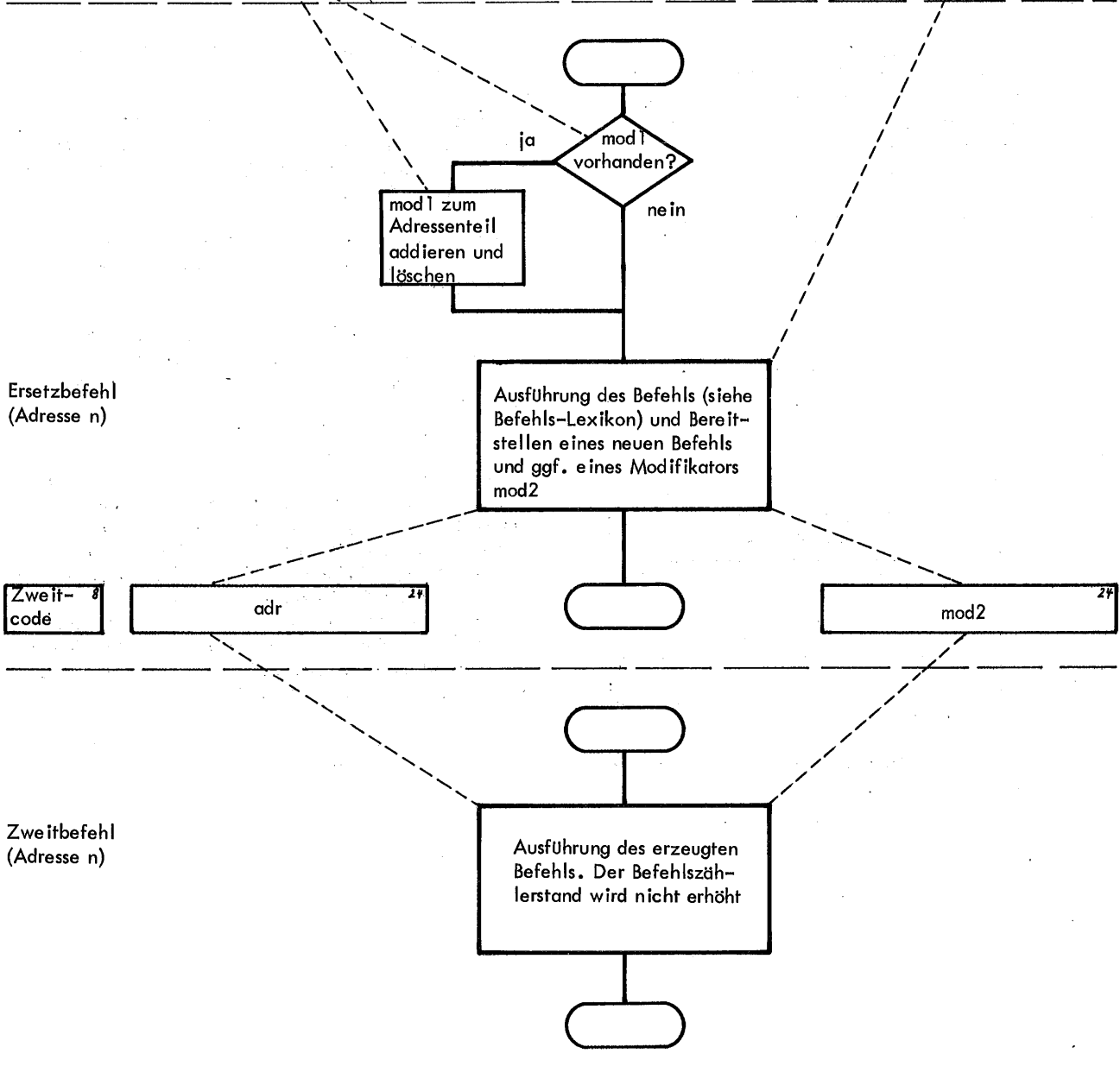
Bild 9.2 siehe umseitig

M

vorhergehender Befehl
(Adresse n-1)

mod1 2#

mod2 2#



folgender Befehl
(Adresse n+1)

Bild 9.2 Ausführung eines Doppelcodebefehls

10. ERSETZBEFEHLE

Gliedert man Ersetzbefehle danach, wie der Adressenteil des Zweitbefehls entsteht, so erhält man folgende Gruppen

Adressenteil aus Indexzelle	E, EZ, ENZ, EMB
Adressenteil aus Register B	MAB, MABI
Adressenteil über Unterprogrammregister	MU, EMU
Adressenteil relativ zum Befehl	RLR
Operand aus Register	R
Tue-Befehl	T

Die Ersetzbefehle (ausgenommen die Befehle R und T) bringen im Adressenteil des Zweitbefehls stets eine 24 Bit-Größe. Damit können Befehle, die im Adressenteil ein n oder m haben, als Zweitbefehle den gesamten Kernspeicher erreichen.

Ist der Zweitbefehl jedoch ein Sprungbefehl, so gilt das, was allgemein für Sprungbefehle in bezug auf Sprung in eine andere Großseite gilt. Für die beiden Ersetzbefehle MABI und MU gilt jedoch, daß auch ein Sprung in eine andere Großseite möglich ist. Näheres zu den Sprüngen siehe unter Abschnitt "Sonstige Operationen".

Zweitbefehl ist der Befehl B, sein Adressenteil wird der Inhalt der Indexzelle X3. Da der Befehl B auf 2. Art modifiziert wird, d.h. also, daß der mod2 zum Adressenteil addiert wird, ergibt sich der Adressenteil des Zweitbefehls aus der Summe der Inhalte der beiden Indexzellen X1 und X3. Wenn wir annehmen, daß

Inhalt von X1 = 1000
Inhalt von X3 = 100

so ergibt sich für die Ausführung

$$\left. \begin{array}{l} M \ X1, \\ E \ B \ X3, \end{array} \right\} \cong B \ 1100,$$

Im zweiten Beispiel steht in der Indexzelle X3 die Adresse des Befehls, auf den gesprungen werden soll. Steht in der Indexzelle der Wert 2000, so wird dieser Befehl ausgeführt als "S 2000". Dies bedeutet, daß in der Großseite 1 auf den Befehl mit der Adresse 2000 gesprungen wird. (beim Befehl S ist diese Sprungadresse wie bei den meisten Befehlen relativ zur Anfangsadresse zur Großseite 1).

M	X1,	
E	B	X3,
E	S	X3,

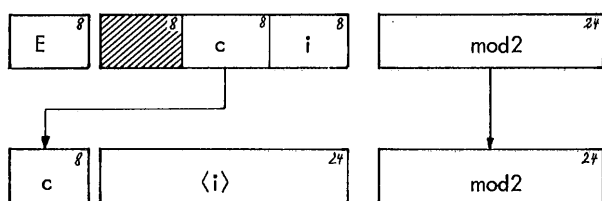
Bild 10.1 Beispiele für Befehl E

10.1. Adressenteil aus Indexzelle

Bei den nachfolgenden Ersetzbefehlen wird der Adressenteil des Zweitcodes aus der in der rechten Hälfte des Adressenteils angegebenen Indexzelle geholt. Der Befehl E ist der einfachste.

E	c	i	Ersetze	op := c adr := <i> mod2 := mod2
---	---	---	---------	---------------------------------------

Ein vom vorhergehenden Befehl vorhandener Modifikator 2. Art wird an den Zweitbefehl weitergegeben, d.h. nicht der Ersetzbefehl wird durch mod2 modifiziert, sondern der Zweitbefehl.

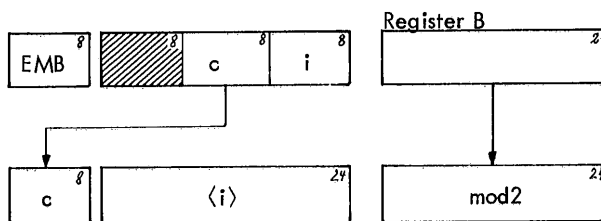


Beim ersten Beispiel in Bild 10.1 wird ein Modifikator 2. Art (mod2) gebildet aus dem Inhalt der Indexzelle X1. Dieser Modifikator modifiziert den Zweitbefehl. Der

Eine Erweiterung des E Befehls bildet der Befehl EMB.

EMB	c	i	Ersetze und modifiziere mit <u>B</u>	op := c adr := <i> mod2 :=
-----	---	---	--------------------------------------	--------------------------------------

Er schließt eine Modifizierung 2. Art mit ein. Als Modifikator (mod2) wird der Inhalt des Registers B verwendet. Vor der Ausführung des Befehls EMB muß also die Modifikationsgröße, mit der der Zweitbefehl modifiziert werden soll, im Register B stehen.



Dieser Befehl ist daher dann angebracht, wenn die Modifikationsgröße auf Grund eines früheren Befehls bereits (oder noch) im Register B steht.

Muß die Größe jedoch erst mit einem Befehl ins Register B gebracht werden, so ist im allgemeinen der Befehl E mit einem vorhergehenden Modifizierbefehl vorzuziehen.



Wird der Befehl EMB auf 2. Art modifiziert, so hat er die gleiche Wirkung wie der Befehl E, da bei allen Modifizierbefehlen die einen mod2 erzeugen, dieser auch im Register B steht.

```

:
HXP 5 X1,
EMB B X2,
EMB A X3,
EMB C X4,
:

```

Bild 10.2 Beispiele für Befehl EMB innerhalb einer Schleife

Im Bild 10.2 wird mit Hilfe des Befehls HXP die Indexgröße um 5 hochgezählt. Sie steht dann gleichzeitig im Register B. Der Befehl EMB verändert das Register B nicht, so daß bei allen drei EMB-Befehlen der Inhalt des Registers gleich ist und somit alle diese Zweitbefehle mit der gleichen Größe modifiziert werden. Das Beispiel von Bild 8.2 wurde im Bild 10.3 auf den Befehl EMB abgewandelt. Eine Zeitberechnung für die Schleife ergibt (nach der "Großen Befehlsliste")

	Bild 8.2	Bild 10.3
Abrufphase	56 Takte	40 Takte
Modifizierung	24	24
Operand aus Speicher erfüllter Sprung	16	16
Ausführungsphase	5	5
	21,5 (10)	31,5 (10)
	122,5 Takte	116,5 Takte

(...) Überlappung der Befehls- und Rechenwerkszeiten

Die Schleife im Beispiel 10.3 braucht also etwas weniger Zeit. Das zusätzliche Setzen der Indexzellen benötigt 60 Takte. Es ist jedoch nur einmal nötig und ggf. schon von vorhergehenden Operationen vorhanden, während der Unterschied in der Schleife sich bei einer großen Anzahl von Durchläufen sehr bemerkbar machen kann.

```

:
XBAN 200,
XC X1,
XBA SUM1+190,
XC X2,
XBA SUM2+190,
XC X3,
XBA SUM+190,
XC X4,
ANF= HXP 10 X1,
EMB B X2,
EMB A X3,
EMB C X4,
SXN ANF,
:

```

Bild 10.3 Beispiele für Befehl EMB

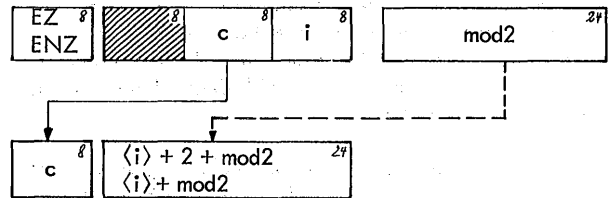
Für den Fall, daß Datenbereiche Wort für Wort durchgearbeitet werden sollen, sind die Befehle EZ und ENZ von Vorteil.

EZ	c i	Ersetze <u>z</u> ählend	op := c adr := <i>+2+mod2 := <i>+2 <i> := <i>+2
ENZ	c i	Ersetze <u>n</u> egativ <u>z</u> ählend	op := c adr := <i>+mod2 := <i>-2 <i> := <i>-2

Außer der Wirkung des Befehls E wird beim Befehl EZ zusätzlich der Inhalt der Indexzelle, also die Adresse für den Zweitbefehl, um ein Ganzwort hochgezählt. Der Zweitcode erhält die um 2 erhöhte Adresse.

Beim Befehl ENZ wird dagegen um 2 heruntergezählt. Hier ist zu beachten, daß der Inhalt der Indexzelle zwar um 2 erniedrigt wird, der Zweitcode aber die nicht erniedrigte Adresse erhält.

Bei beiden Befehlen ist der Inhalt der Indexzelle gleichzeitig im Register B. Ein vom vorhergehenden Befehl vorhandener Modifikator 2. Art wird in jedem Falle unabhängig vom Zweitcode zum Adressenteil des Zweitbefehls addiert.



Diese beiden Ersetzbefehle sind besonders für Befehle geeignet, die Ganzwörter oder Halbwörter ansprechen. Innerhalb einer Schleife können aufeinanderfolgende Wörter (bzw. jedes zweite Halbwort) angesprochen werden. Die Indexzelle i muß in diesem Fall enthalten:

- bei EZ Adresse des ersten Wortes minus 2
- bei ENZ Adresse des letzten Wortes

Im Beispiel Bild 10.4 wird aus dem Inhalt von 10 Ganzwörtern, die in einem Block stehen, der bei SUM beginnt, die Summe gebildet. Dabei wird mit dem letzten Wort begonnen. Erst wenn die Indexzelle X1 auf den Wert -2 gekommen ist, wird die Schleife verlassen, d.h. solange sie gleich oder größer 0 ist wird gesprungen.

```

:
ZX 18 X1,
LR A1,
MA SUM,
ENZ A X1,
SXGG -2R,
:

```

Durchlauf	Inhalt von X1	Adresse bei Befehl A
1.	18 - 2 = 16	SUM + 18
2.	16 - 2 = 14	SUM + 16
usw.		
9.	2 - 2 = 0	SUM + 2
10.	0 - 2 = -2	SUM + 0

Bild 10.4 Beispiel für Befehl ENZ

Im Beispiel Bild 10.5 ist eine Variante des Beispiels in Bild 10.3 gezeigt. Hier werden jedoch jeweils aufeinanderfolgende Wörter in den Blöcken angesprochen. Es wurden jeweils 10 Wörter miteinander verknüpft.

```

:
XBA SUM1-2,
XC X2,
XBA SUM2-2,
XC X3,
XBA SUM-2,
XC X4,
ANF= EZ B X2,
EZ A X3,
EZ C X4,
VBA SUM+18,
SXN ANF,
:

```

$$\langle \text{SUM1+a} \rangle + \langle \text{SUM2+a} \rangle = \langle \text{SUM+a} \rangle$$

$$a = 0, 2, 4, \dots, 16, 18.$$

Durchlauf	Inhalt von X1	Adresse bei Befehl C
1.	SUM - 2 + 2 =	SUM
2.	SUM + 2 =	SUM + 2
usw.		
9.	SUM +14 + 2 =	SUM + 16
10.	SUM +16 + 2 =	SUM + 18

Bild 10.5 Beispiel für Befehl EZ

```

:
ZX -2 X1,
XBA SUM1,
XC X2,
XBA SUM2,
XC X3,
XBA SUM,
XC X4,
ANF= M X2,
EZ B X1,
M X3,
E A X1,
M X4,
E C X1,
XB X1,
VBA 18,
SXN ANF,
:

```

$$\langle \text{SUM1+a} \rangle + \langle \text{SUM2+a} \rangle = \langle \text{SUM+a} \rangle$$

$$a = 0, 2, 4, \dots, 16, 18.$$

Durchlauf	Inhalt von X1	Adresse von Befehl C
1.	-2 + 2 = 0	SUM
2.	0 + 2 = 2	SUM + 2
3.	2 + 2 = 4	SUM + 4
usw.		
9.	14 + 2 = 16	SUM + 16
10.	16 + 2 = 18	SUM + 18

Bild 10.6 Beispiel für Befehl EZ, E und M

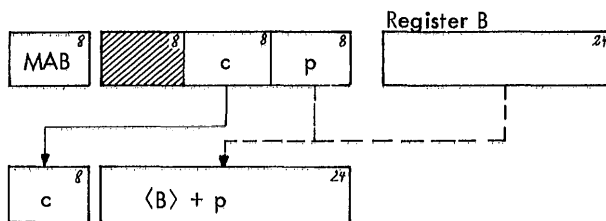
In Bild 10.6 ist der gleiche Fall wie in Bild 10.5 nur ist hier mit den Befehlen E, EZ und M gearbeitet worden.

10.2. Adressteil aus Register B.

Der Adressteil des Zweitbefehls kann mit Hilfe des Befehls MAB auch dem Register B entnommen werden.

MAB	c	p	Modifiziere Adressteil mit Register B	op := c adr := + p := + p p: ±0...±127
-----	---	---	---------------------------------------	---

Zu dem Adressteil wird noch die Zahl p hinzugezählt. Ist dies nicht erwünscht, so kann p = 0 gesetzt werden. Die Zahl p kann die Werte ±0 bis ±127 annehmen, d.h. es kann maximal 127 addiert oder subtrahiert werden.



Eine Modifizierung 2. Art kann nur insofern einen Sinn haben, daß der mod2 stets auch im Register B steht und damit dann mod2+p Adressteil des Zweitbefehls wird.

```

:
XBA DATEN,
XC X1,
ANF= XB X1,
MAB B 4,
MAB ML 2,
MAB CQ 2,
HXP 10 X1,
VBA DATEN+100,
SXN ANF,
:

```

$$\langle \text{DATEN+a+4} \rangle \cdot \langle \text{DATEN+a+6} \rangle = \langle \text{DATEN+a+8} \rangle$$

$$a = 0, 10, 20, \dots, 80, 90.$$

Durchlauf	Inhalt von X1	Adresse von Befehl		
		B	ML	C
1.	DATEN	DATEN +4	DATEN +6	DATEN +8
2.	DATEN+10	DATEN+14	DATEN+16	DATEN+18
usw.				
9.	DATEN+80	DATEN+84	DATEN+86	DATEN+88
10.	DATEN+90	DATEN+94	DATEN+96	DATEN+98

Bild 10.7 Beispiel zum Befehl MAB

Im Beispiel Bild 10.7 ist angenommen, daß beginnend bei der Adresse DATEN 50 Wörter abgelegt sind. Sie sind in 10 Blöcke à 5 Wörter aufgeteilt. Innerhalb jedes Blockes wird das 3. Wort (<X1>+4) mit dem 4. Wort (<X1>+6) multipliziert und das Ergebnis in das 5. Wort (<X1>+8) gebracht. Der Inhalt der Indexzelle X1 wird durch den Befehl HXP hochgezählt. Sie enthält immer die Adresse des ersten Wortes eines Blockes. Nach dem



Befehl XB steht der Inhalt der Indexzelle X1 im Register B. Durch den ersten Befehl MAB wird diese Adresse um 4 hochgezählt auf $\langle X1 \rangle + 4$ (3. Wort im Block), durch den zweiten um 2 auf $\langle X1 \rangle + 6$ (4. Wort im Block) und durch den dritten um 2 auf $\langle X1 \rangle + 8$ (5. Wort im Block).

Die gleiche Wirkung wie der Befehl MAB hat der Befehl MABI.

MABI	c	p	Modifiziere Adressenteil mit Register B bei Invarianz der Sprungadresse	op := c adr := $\langle B \rangle + p$ $\langle B \rangle := \langle B \rangle + p$
------	---	---	---	---

Lediglich wenn der Zweitcode ein Sprungbefehl ist, gilt für diesen Sprungbefehl, daß er in jede andere Großseite springen kann. Es werden also bei erfüllter Sprungbedingung (und bei unbedingten Sprüngen) alle 24 Bits des Adressenteils in das Befehlsregister übertragen.

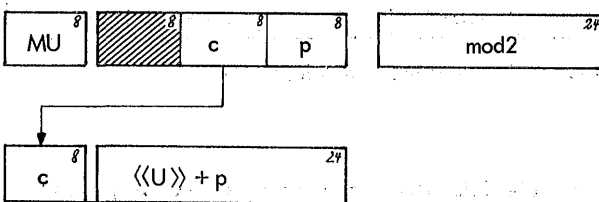
10.3. Adressenteil über Unterprogrammregister

Die beiden Ersetzbefehle MU und EMU werden nur innerhalb von Unterprogrammen verwendet, um Halbwörter zu erreichen, die in übergeordneten Programmen (Hauptprogramm) hinter bzw. vor dem Befehl SU oder SUE (Sprung ins Unterprogramm) stehen. Nähere Angaben zur Programmorganisation sind dem Abschnitt "Anschluß anderer Programme" zu entnehmen.

MU	c	p	Modifiziere über Register U	op := c adr := $\langle \langle U \rangle \rangle + p$
EMU	c	p	Ersetze nach Modifizierung über Register U	op := c adr := $\langle \langle \langle U \rangle \rangle + p \rangle$ mod2 := mod2

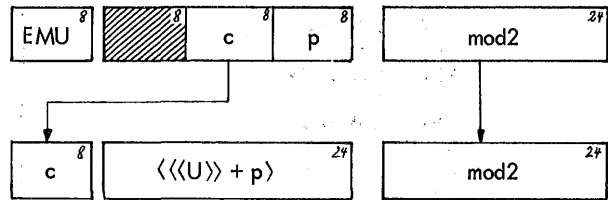
p: $\pm 0 \dots \pm 127$

Beim Sprung ins Unterprogramm (Befehl SU oder SUE) wird die auf diesen Befehl folgende Adresse (technische Rücksprungadresse) in einer Indexzelle sichergestellt. Die Adresse dieser Indexzelle steht im Unterprogrammregister U. Mit dem Befehl MU wird (innerhalb des Unterprogramms) ein Zweitbefehl erzeugt, der im Adressenteil die auf den (im übergeordneten Programm stehenden) Befehl SU bzw. SUE folgende Adresse (technische Rücksprungadresse) enthält. Der Adressenteil kann noch um den Wert p erhöht werden, wobei p die Werte ± 0 bis ± 127 annehmen kann. Dies bedeutet, daß 126 Halbwörter vor und 128 Halbwörter hinter dem Befehl SU bzw. SUE erreicht werden können. Eine Modifizierung zweiter Art ist nicht möglich.



Während der Befehl MU also die Adresse einer der Zellen vor oder hinter dem Befehl SU bzw. SUE dem Zweitbefehl zur Verfügung stellt, wird vom Befehl EMU nochmals eine Ersetzung vorgenommen und der

Inhalt dieses Halbwortes als Adressenteil des Zweitbefehls verwendet. Der Adressenteil des Zweitbefehls ist also der Inhalt eines Halbwortes das (in übergeordneten Programmen) hinter oder vor dem Befehl SU bzw. SUE steht.



Ist ein Modifikator 2. Art vom vorhergehenden Befehl vorhanden, so wird er an den Zweitbefehl weitergegeben, d.h. bei der Modifizierung 2. Art wird nicht der Ersetzbefehl modifiziert, sondern der Zweitbefehl.

Der Befehl MU wird benötigt, um von Unterprogramm in das übergeordnete Programm zurückzuspringen. In diesem Fall ist der Zweitbefehl ein Sprungbefehl und wenn $p=0$ ist, so wird bei einem unbedingten Sprung bzw. bei erfüllter Sprungbedingung ins übergeordnete Programm zurückgesprungen und zwar hinter den Befehl SU bzw. SUE (technische Rücksprungadresse). Mit Hilfe des Wertes p kann auch auf einen Befehl gesprungen werden, der um p Stellen hinter bzw. vor der technischen Rücksprungadresse liegt.

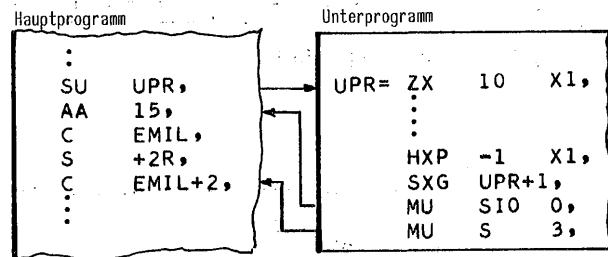


Bild 10.8 Beispiel Rücksprung ins Hauptprogramm

Ist bei MU der Zweitbefehl ein Sprungbefehl, so ist (wie bei MABI) ein Sprung in eine andere Großseite möglich.

Mit Hilfe der Befehle MU und EMU ist es auf einfache Weise möglich, Daten, die für das Unterprogramm bestimmt sind, in Form von Versorgungsblöcken hinter oder auch vor den Befehl SU bzw. SUE zu schreiben. Die Befehle sind im Normalfall schreibgeschützt. In diesem Fall sind dann auch die Daten des Versorgungsblocks schreibgeschützt. Es muß dafür gesorgt werden, daß die Daten des Versorgungsblocks nach der Übersetzung auch mit im Befehlsbereich stehen. Konstanten müssen also mit der Spezifikation B versehen werden.

Im Beispiel Bild 10.10 werden vom Unterprogramm eine Anzahl Zahlen mit einer Konstanten multipliziert und das Ergebnis abgespeichert. Dazu müssen im Versorgungsblock angegeben werden: Anfangsadresse der Zahlen (MULT), die Konstante (70000), Anfangsadresse für die Ergebnisse (ERGEB) und die Anzahl der Zahlen in Halbwörtern ($50 \times 2 = 100$).

Da die Daten des Versorgungsblocks innerhalb der Befehle stehen, jedoch keine Befehle sind, muß dafür gesorgt

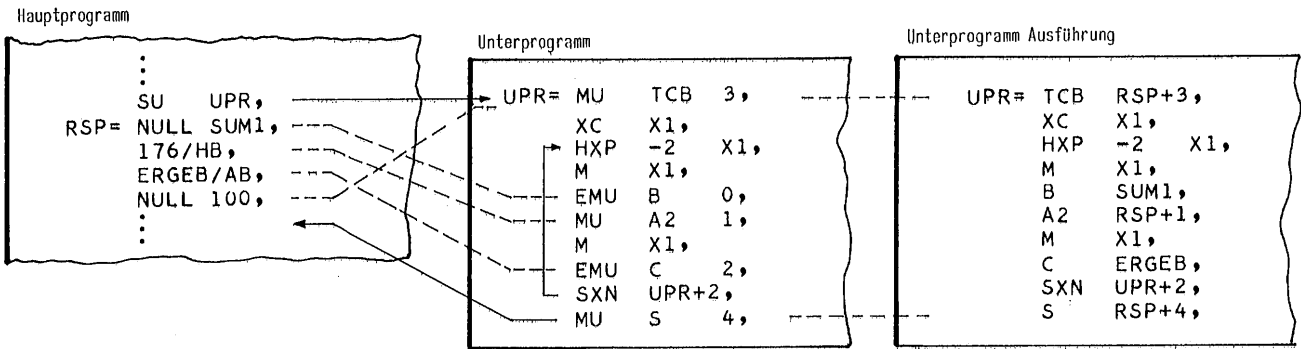


Bild 10.9 Beispiel für die Befehle MU und EMU

werden, daß sie bei dem sequentiellen Ablauf des Programms nicht ins Befehlswerk geholt werden. Stehen die Versorgungsdaten hinter dem Befehl SU bzw. SUE, so muß der Rücksprung hinter diese Daten erfolgen. Stehen sie vor dem Befehl, so können sie, solange sie nicht länger als 16 Bit sind, im Adressenteil des Befehls NULL stehen (1. Beispiel im Bild 10.10). Dieser Befehl hat keine Wirkung. Sind sie größer als 16 Bit, so müssen sie entsprechend übersprungen werden (2. Beispiel im Bild 10.10). Die Versorgungsdaten können auch mehrfache Wortlänge haben.

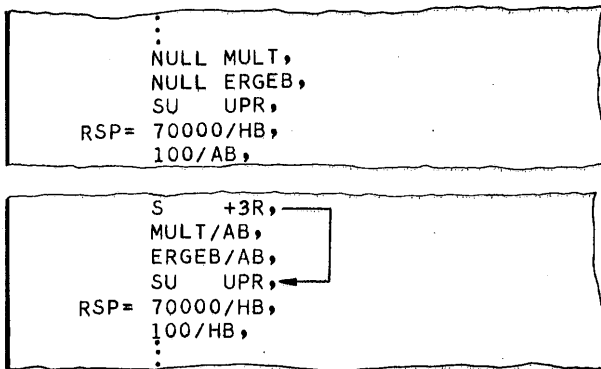


Bild 10.10 Beispiele für Versorgungsblöcke

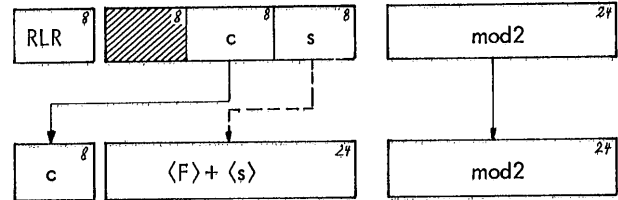
10.4. Adressenteil relativ zum Befehl

Der Befehl RLR ermöglicht eine Adressierung relativ zu seiner eigenen Adresse.

RLR	c	s	Relativ-Adressierung mit Registerinhalt	op := c adr := <s> + <F> mod2 := mod2
-----	---	---	---	---

c: Code des Zweitbefehls
s: Register A, Q, D, H (rechte Hälfte)
AL, QL, DL, HL (linke Hälfte)
F, B, U, Y

Im Adressenteil des Zweitbefehls steht der Inhalt des Registers F - dies ist die Adresse des Befehls RLR - erhöht um den Inhalt des mit der Spezifikation s angegebenen Registers.



Bei den Rechenwerkregistern wird die rechte Hälfte adressiert. Soll die linke Hälfte adressiert werden, so ist dem Registerbuchstaben der Buchstabe L hinzuzufügen. Bei der Angabe des Registers U und Y werden die Inhalte dieser Register (auf die rechten 8 Bits) adressiert. Ein vom Vorbefehl vorhandener Modifikator 2, Art (mod2) wirkt auf den Zweitbefehl.

Mit Hilfe des Befehls RLR kann ein Sprung relativ zum Befehl RLR durchgeführt werden. Um wieviel Befehle vor bzw. zurückgesprungen wird, steht dann in dem mit s angegebenen Register.

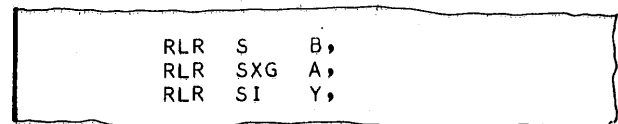


Bild 10.11 Beispiele zum Befehl RLR

10.5. Operand aus Register

Mit Hilfe des Befehls R ist es möglich, mit einem Befehl, der einen Operanden aus dem Speicher holt, den Operanden nicht aus dem Speicher, sondern aus einem Register zu holen.

R	c	s	Registeradressierung	op := c operand := <s ₁ >
---	---	---	----------------------	---

s₁: Register A, Q, D, H,
Y, U,
B oder F

s₂: leer = rechte Hälfte } nur wenn Zweitbefehl
L = linke Hälfte } ein Halbwort adressiert

Zu diesem Zwecke wird der Befehl, der den Operanden aus einem Register A, Q, D, H, F, B, U oder Y holen soll, als Zweitbefehl beim Befehl R eingetragen. Der Zweitbefehl läuft dann so ab, als ob der Inhalt des angesprochenen Registers im Speicher stünde. Ein vom vorhergehenden Befehl vorhandener Modifikator wird zum Adressenteil des Befehls R addiert.

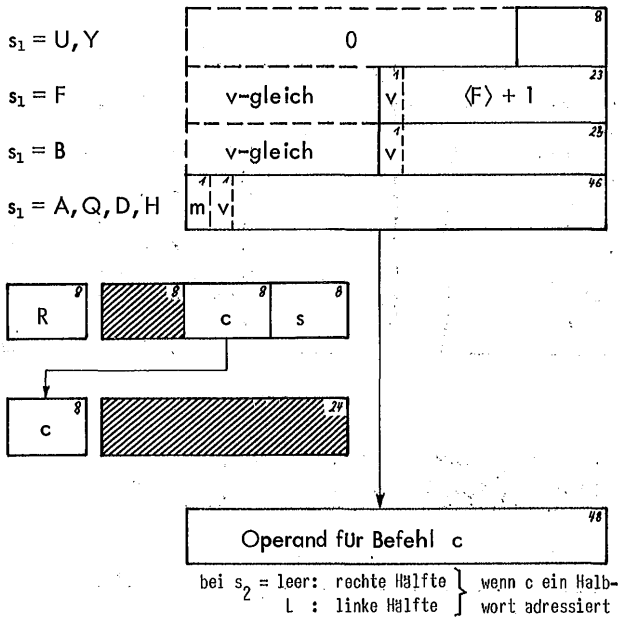


Bild 10.12 Wirkung des Befehls R

Als Zweitbefehl sind nur die Befehle zugelassen, die ein einziges Ganzwort oder Halbwort aus dem Speicher holen. In Bild 10.13 sind diese Befehle ausgeführt.

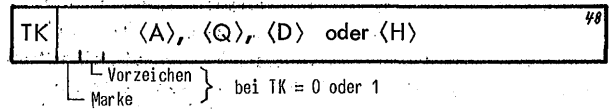
- A A2 AB AQ AT AU AUT
- B B2 B2V B2VN B3 B3V BB BD BH BN
- BNR BQ BQB BR BT BU
- DV DVD DVI
- ET
- GA GAB GDV GDVI GMAN GML GMLA
- GMLN GSB GSBB GSBD GSBI
- HBC
- M2 M2N M2NR M2R MAN MANR MAR
- MC MCE MCF MCFU ML MLA MLN MLR
- MNR
- NULL
- REZ
- SB SB2 SBB SBD SBI SBQ SBT SBU SE
- SUE
- T TCB
- VBC VEL
- ZUS

Bild 10.13 Zulässige Zweitcodes beim Befehl R

Der Zweitbefehl kann ein Ganzwort adressieren oder ein Halbwort. Wird ein Ganzwort adressiert, so wird entsprechend der Wirkung des Befehls der Operand verarbeitet. Wird ein Halbwort adressiert, so wird die rechte Hälfte des Operanden entsprechend der Wirkung des Befehls verarbeitet. Soll die linke Hälfte verwendet werden, so ist zu der Angabe des Registers der Buchstabe L hinzuzufügen.

Der Operand wird stets so aufgebaut, als ob er im Speicher stünde. Er hat also neben der Typerkennung noch 48 Binärstellen. Er wird bei den einzelnen Registern wie folgt gebildet.

Register A, Q, D, H: Das Wort aus den Registern wird unverändert als Operand übernommen. Bei Zahlwörtern ist jedoch zu beachten, daß sie so behandelt werden, als ob sie mit dem Befehl BU gespeichert wären. Es ist also - wie im Speicher - das Bit Nummer 1 die Marke des Operanden und Bit Nummer 2 das Vorzeichen des Operanden. Ein negatives Zahlwort ist also markiert, ein positives nicht. Ein (positives) übergelaufenes Zahlwort ist nicht markiert und negativ und ein (negatives) untergelaufenes Zahlwort ist positiv und markiert. Zweckmäßigerweise sollten aber keine über- oder unterlaufenen Zahlwörter verarbeitet werden (siehe Bild 10.14).



Zahlwort im Register	Bitmuster	Zahlwort als Operand
positiv	00...	positiv nicht markiert
positiv übergelaufen	0L...	negativ nicht markiert
negativ	LL...	negativ markiert
negativ untergelaufen	LO...	positiv markiert

Bild 10.14 Operand aus Register A, Q, D oder H

Register B, F: Das Bit Nummer 1 beim Register B bzw. F ist das Vorzeichen. Der Inhalt des Registers ergibt die rechte Hälfte des Operanden. Die linke Hälfte wird dem Vorzeichen gleichgesetzt. Das bedeutet, daß der Inhalt des Registers B bzw. F vorzeichenrichtig als Operand erscheint. Der Operand erhält die Typenkennung 1. Da bei Zahlwörtern das 1. Bit das Markenbit ist, sind also bei negativem Inhalt des Registers B bzw. F das Wort markiert.

Das Register F enthält die Adresse des Befehls R um 1 erhöht.

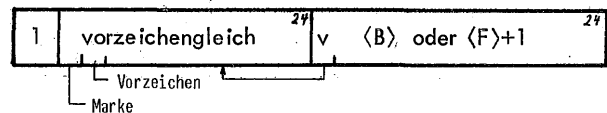


Bild 10.15 Operand aus Register B oder F

Register Y, U: Der Inhalt des Registers Y bzw. U ergibt die rechten 8 Bit des Operanden. Die linken 40 Bit werden auf Null gesetzt. Der Operand erhält die Typenkennung 1.

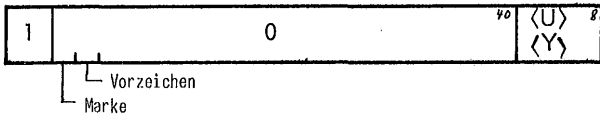


Bild 10.16 Operand aus Register U oder Y

Mit Hilfe des Befehls R ist es möglich

- den Inhalt eines Registers in ein anderes zu bringen
- den Inhalt zweier Register miteinander zu verknüpfen
- auf die in einem Register gegebene Adresse zu springen
- mit dem Inhalt eines Registers zu modifizieren
- einen in einem Register stehenden Befehl auszuführen
- den Inhalt des Registers Y zu erreichen

Die im Befehl R angegebenen Register werden dabei durch den Befehl R nicht verändert. Eine Veränderung durch den Zweitcode hängt nur von dessen Wirkung ab.

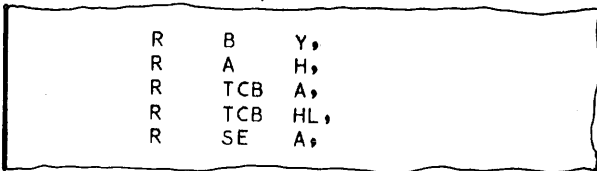


Bild 10.17 Beispiele für den Befehl R

Mit Hilfe des Befehls ist es auf einfache Art möglich, einen Befehl in einem anderen Bereich als den B-Bereich zu bringen. An seine Stelle tritt der T-Befehl. Der Programmablauf wird damit nicht verändert. Liegt der Befehl z.B. im V-Bereich, so ist er nicht schreibgeschützt und kann per Programm verändert werden.

In Verbindung mit dem Befehl R ist es möglich, in einem Register einen Befehl aufzubauen und diesen Befehl anstelle des T-Befehls in das Programm einzubauen.

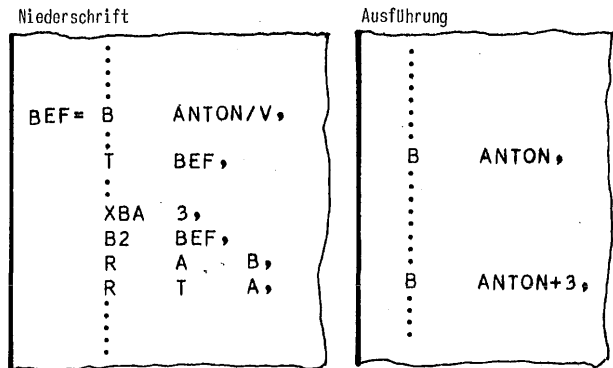


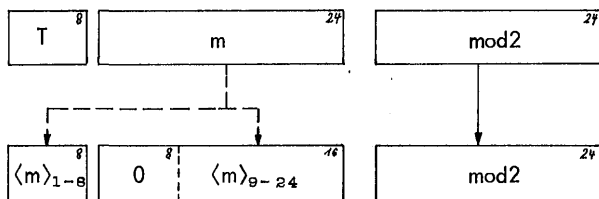
Bild 10.18 Beispiele für Befehl T

10.6. Befehl T

Der Befehl T zählt zu den Ersetzbefehlen (Doppelcodebefehlen) hat jedoch im Adressenteil die Adresse eines Ganzwortes.



In der durch m angegebenen Speicherzelle steht ein Befehl. Er braucht nicht die Typenkennung 2 zu haben. Dieser Befehl wird der Zweitbefehl und anstelle von T ausgeführt.



Ohne Modifizierung kann nur ein Befehl erreicht werden, der in der Großseite 0 (K- und V-Bereich) liegt. Durch eine Modifizierung 1. Art kann der Befehl in jeder Großseite erreicht werden. Eine Modifizierung 2. Art wirkt nicht auf den Befehl T sondern auf den Zweitbefehl.



11. AUFBAU VON SCHLEIFEN

Um Schleifen innerhalb von Programmen aufzubauen, bedarf es eines Abbruchkriteriums. Die Schleife muß so oft durchlaufen werden, bis das Abbruchkriterium erreicht ist. Zur Abfrage des Abbruchkriteriums werden die bedingten Sprungbefehle verwendet, Auf Grund der Sprungbedingung dieser Befehle können verglichen werden

- Register B mit 0
- Indezzelle mit 0
- Register A mit 0
- Register A mit Register H
- Exponent im Register A

Des weiteren können abgefragt werden

- jede Binärstelle in den Registern A, Q, D und H,
- Typenkennung im Register A, Q, D und H,
- Merklichter
- Markenregister
- BÜ und TK-Alarm

ob sie einen bestimmten Wert haben. Ist die zur Wirkung des Befehls gehörende Sprungbedingung erfüllt, so wird gesprungen.

Nachstehend sind einige Möglichkeiten der Schleifenbildung aufgezeichnet, gegliedert nach den Abbruchkriterien. Die Sprungbefehle werden hier nicht erklärt, Sie sind unter "Sonstige Operationen" beschrieben.

```

      :
ZGR= ZX  0  X1,
ANF= LR  A1,
      :
      HXP  1  X1,
      VBA 100,
END= SXN ANF,
      :
  
```

Zählgröße auf 0 setzen

Zählgröße erhöhen
Vergleich der Zählgröße mit 100

Durchlauf	Zählgröße		Reg. B nach Subtr.	Sprungbed. erfüllt
	am Anf.	am Ende		
1.	0	1	-99	ja
2.	1	2	-98	ja
3.	2	3	-97	ja
usw.				
99.	98	99	-1	ja
100.	99	100	0	nein

Bild 11.1 Beispiel für Zählgröße auf Null setzen

11.1. Zählgröße

Ist bekannt, wie oft eine Schleife durchlaufen werden soll, so kann das Abbruchkriterium eine Zählgröße sein.

Im Beispiel Bild 11.1. wird die Schleife, die bei ANF beginnt und bei END endet, 100 mal durchlaufen. Während des ersten Durchlaufs hat sie den Wert 0. Am Ende der Schleife wird sie um 1 erhöht, hat also den Wert 1. Dieser Wert steht in der Indezzelle X1, aber auch im Register B. Das Register B kann jedoch nur mit Null verglichen werden. Es wird also der gewünschte Maximalwert (100 Durchläufe) vom Register B subtrahiert. Der Inhalt des Registers B wird im ersten Durchlauf nach der Subtraktion also den Wert -99 haben, er ist also nicht identisch Null. Die Schleife wird erneut durchlaufen. Am Anfang des zweiten Durchlaufs hat die Zählgröße den Wert 1, am Ende den Wert 2, und der Inhalt des Registers B ergibt sich nach der Subtraktion zu -98. Im Bild 11.1. ist die Zählgröße und der Inhalt des Registers B in Abhängigkeit von den Durchläufen angegeben.

Erst beim 100sten Durchlauf wird das Register B nach der Subtraktion den Wert 0 haben. Damit ist die Sprungbedingung nicht mehr erfüllt, und die Schleife wird verlassen. Faßt man die beiden Befehle VBA 100 und SXN ANF zusammen, so kann man sagen: wenn die Zählgröße nicht den Wert 100 hat wird die Schleife erneut durchlaufen (wird gesprungen).

Es ist zu beachten, daß die Zählgröße (am Anfang und am Ende) 100 verschiedene Werte annimmt. Am Anfang sind 0 bis 99 (= 100 Zahlen), und am Ende sind es 1 bis 100 (= 100 Zahlen).

```

      :
ZGR= ZX  100 X1,
ANF= LR   A1,
      :
      HXP -1  X1,
END= SXN ANF,
      :
  
```

Durchlauf	Zählgröße		Reg. B nach Subtr.	Sprungbed. erfüllt
	am Anf.	am Ende		
1.	100	99	99	ja
2.	99	98	98	ja
3.	98	97	97	ja
usw.				
99.	2	1	1	ja
100.	1	0	0	nein

Bild 11.2 Beispiel Zählgröße auf Maximalwert setzen



TAS - Handbuch

Dez. 68

Im vorstehenden Fall wurde die Zählgröße auf 0 gesetzt und dann hochgezählt. Man kann sie jedoch auch auf den Maximalwert setzen und herunterzählen. Auf diesem Wege erreicht sie beim letzten Durchlauf den Wert 0 und eine Subtraktion des Maximalwertes ist nicht erforderlich. Die Schleife wird also um einen Befehl gekürzt (siehe Bild 11.2).

In diesem Fall wäre es günstiger, das Herunterzählen an den Anfang der Schleife zu legen (siehe Abschnitt 11.2.1.).

Die Zählgröße kann auch auf den negativen Maximalwert gesetzt werden. Im Beispiel Bild 11.3 ist dann die Zählgröße hochzuzählen.

Ein besonders einfacher Schleifenaufbau ergibt sich mit dem Befehl SZX (siehe Bild 11.3.).

```

ZGR= ZX  ~99  X1,
A=   LR   A1,
.
.
E=   SZX  A-E  X1,
.
.

```

Bild 11.3 Beispiel Schleife mit SZX

Der Befehl SZX ist ein Sprungbefehl mit einer relativen Sprungadresse. Im Adressenteil steht die Anzahl der Befehle um die vorwärtsgesprungen wird (positive Zahl) bzw. um die zurückgesprungen wird (negative Zahl). Ist, wie im Beispiel 11.3., die Sprungweite nicht bekannt, so kann auch eine Differenzadresse (im Beispiel A-E) angegeben werden. Vom Assembler wird die Differenz errechnet und eingesetzt. Wird die Sprungweite in dieser Art angegeben, so können bei notwendigen Korrekturen Befehle eingeschoben oder fortgelassen werden, ohne daß die Sprungweite geändert werden muß.

Im Beispiel Bild 11.3 wurde die Indezelle X1 auf den Wert -99 gesetzt. Damit wird die Schleife 100 mal durchlaufen.

Ist die Zahl der Durchläufe größer als 127, so kann die Zählgröße nicht mehr mit dem Befehl ZX gesetzt werden. Setzen von Indezellen ist im Abschnitt 2 beschrieben. Beim Befehl VBA liegt die Grenze bei $2^{16} - 1$.

11.2. Zählgröße gleichzeitig Modifiziergröße

Bei einem großen Teil der Schleifen wird neben der Zählgröße noch eine Modifiziergröße benötigt, mit der die in der Schleife verwendeten Adressen hochgezählt werden können.

11.2.1. Schrittweite 1

Werden in der Schleife aufeinanderfolgende Halbwörter angesprochen, so kann Zählgröße und Modifiziergröße identisch sein.

```

.
.
ZGR= ZX  0    X1,
ANF= M   X1,
     B2   ZAHL,
.
.
     HXP  1    X1,
     VBA 100,
END= SXN ANF,
.
.

```

Durchlauf	Zählgröße		Register B nach Subtr.	Adresse von B2
	am Anf.	am Ende		
1.	0	1	-99	ZAHL+ 0
2.	1	2	-98	ZAHL+ 1
3.	2	3	-97	ZAHL+ 2
usw.				
99.	98	99	-1	ZAHL+98
100.	99	100	0	ZAHL+99

Bild 11.4 Zählgröße gleich Modifiziergröße

Im Beispiel Bild 11.4. beginnt bei der Adresse ZAHL eine Tabelle von 100 Halbwörtern. Das erste hat die Adresse ZAHL+0 und das letzte die Adresse ZAHL+99. Sie werden nacheinander durch die Schleife ins Register A gebracht und können dort verarbeitet werden.

```

.
.
ZGR= ZX  100 X1,
ANF= HXP -1  X1,
     B2   ZAHL,
.
.
     XB   X1,
END= SXN ANF,
.
.

```

```

.
.
ZGR= ZX  100 X1,
ANF= M   X1,
     B2   ZAHL-1,
.
.
     HXP -1  X1,
END= SXN ANF,
.
.

```

Durchlauf	Zählgröße		Adresse von Befehl B2
	am Anfang	am Ende	
1.	99 (100)	99	ZAHL + 99
2.	98 (99)	98	ZAHL + 98
3.	97 (98)	97	ZAHL + 97
usw.			
99.	1 (2)	1	ZAHL + 1
100.	0 (1)	0	ZAHL + 0

(...) = unteres Beispiel

Bild 11.5 Zählgröße gleich Modifiziergröße auf Maximalwert gesetzt

Soll die Zählgröße auf den Maximalwert gesetzt werden, wie im Bild 11.2., so würden die Wörter aus der Tabelle in einer anderen Reihenfolge gebracht und zwar mit dem letzten Wort beginnend. Außerdem würde als erstes das Wort mit der Adresse ZAHL+100 gebracht, das ja nicht mehr zur Tabelle gehört. Bild 11.5. oberes Beispiel zeigt, daß in dem Fall, wo die Zählgröße auf den Maximalwert gesetzt wird, das Herunterzählen besser an den Anfang der Schleife gelegt wird. In diesem Fall muß ggf. am Ende der Schleife der Befehl XB eingeschoben werden, wenn die Zählgröße nicht von einem vorhergehenden Befehl im Register B steht.

Im Bild 11.5. unteres Beispiel ist gezeigt, daß man auch beginnend am Ende der Schleife, hochzählen kann, wenn von der Adresse des Befehls B eine 1 subtrahiert wird.

11.2.2. Schrittweite 2

Werden in einer Schleife aufeinanderfolgende Ganzwörter (oder jedes zweite Halbwort) angesprochen, müssen die Adressen dieser Wörter bei jedem Schleifendurchlauf um 2 erhöht werden. Wenn man die Zählgröße um jeweils 2 hochzählt und den Maximalwert auf den doppelten Wert setzt, so kann Zählgröße und Modifiziergröße zum Hochzählen der Adresse identisch sein.

In Bild 11.6. ist ein Beispiel angegeben, das die entsprechende Variation des Beispiels aus Bild 11.5. ist.

```

      :
      :
ZGR= XBA 200,
      XC  X1,
ANF= HXP -2  X1,
      M  X1,
      B  ZAHL,
      :
      :
      XB  X1,
      :
  
```

Durchlauf	Zählgröße	Adresse von B
1.	198	ZAHL + 198
2.	196	ZAHL + 196
3.	194	ZAHL + 194
usw.		
98.	4	ZAHL + 4
99.	2	ZAHL + 2
100.	0	ZAHL + 0

Bild 11.6 Beispiel für Schleife mit Schrittweite 2

11.2.3. Schrittweite beliebig

Nach der im Abschnitt 11.2.2. erläuterten Art kann man die Schrittweite beliebig wählen. In einer Reihe von Fällen wird es jedoch angezeigt sein, Zählgröße und Modifiziergröße zu trennen, besonders dann, wenn verschiedene Modifiziergrößen erforderlich sind. In Bild

11.7. ist ein Beispiel angegeben. Die Daten sind in Gruppen von 5 Ganzwörtern zusammengefaßt. Die Zählgröße (Inhalt von X1) wird von -20 heraufgezählt, während die Modifiziergröße von 0 beginnend heraufgezählt wird; es wird am Anfang der Daten begonnen. Durch den Befehl B wird also jedes 5. Wort ins Register A gebracht und kann dort weiterbearbeitet werden.

```

      :
      :
      ZX -20 X1,
      ZX  0  X2,
A =   M  X2,
      B  ZAHL,
      :
      :
      HXP 10  X2,
E =   SZX A-E X1,
      :
      :
  
```

Durchlauf	Zählgröße	Modifiziergröße	Adresse bei Befehl B
1.	-20+1 = -19	0+10 = 10	ZAHL + 0
2.	-19+1 = -18	10+10 = 20	ZAHL + 10
usw.			
19.	-2+1 = -1	180+10 = 190	ZAHL + 180
20.	-1+1 = 0	190+10 = 200	ZAHL + 190

Bild 11.7 Beispiel Zählgröße und Modifiziergröße getrennt

11.3. Rechenwerksregister

Als Abbruchkriterium für eine Schleife kann der Inhalt eines Rechenwerksregisters dienen. Ein Vergleich ist möglich zwischen den Registern A und H und zwischen dem Register A und dem Wert Null. Mit Hilfe des Befehls RT (Registertausch) und mit Hilfe des Ersetzbefehls R (Registeradressierung) können die Inhalte aller Register miteinander bzw. die Inhalte aller Register mit Null verglichen werden.

```

      RT  AQ,
      RT  HD,
      SI  ANF,
      :
      :
      RT  AH,
      SIO ANF,
      :
      :
      R   BU  Q,
      SIO ANF,
  
```

Vergleich Register Q und D auf Identität

Vergleich Register H mit Null

Vergleich Register Q mit Null

Bild 11.8 Vergleich beliebiger Rechenwerksregister

Für den Vergleich stehen die im Bild 11.9. aufgeführten Befehle zur Verfügung, die den Inhalt des Registers A entweder mit Null oder mit dem Inhalt des Registers H vergleichen. Bei Gleitkommazahlen kann der Wert des Exponenten mit einer im Adressenteil des Befehls stehenden Größe auf gleich oder größer verglichen werden. Des Weiteren läßt sich jede Binärstelle und auch die Typenkenntnis in den Registern abfragen. In Bild 11.9. sind die Möglichkeiten in Form einer Tabelle aufgeführt.



TAS - Handbuch

Dez. 68

Vergleich Register A		mit Register H		mit Null	
identisch	=	SI	m	SIO	m
nicht identisch	≠	SN	m	SNO	m
gleich oder größer	≥	SGG	m	SGGO	m
größer	>	SG	m	SGO	m
gleich oder kleiner	≤	SKG	m	SKGO	m
kleiner	<	SK	m	SKO	m

rechtes Bit im Register A	= L	SR	m
	= 0	SRN	m
beliebiges Bit im Register A, Q, D oder H	= L	SBIT	p s

Typenkennung im Register A, Q, D oder H	= s ₁	ST	p s
	≠ s ₁	STN	p s

Markenregister	= L	SM	m
	= 0	SMN	m

TK-Alarm	SAT	m
BÜ-Alarm	SAA	m

Bild 11.9 Vergleichsmöglichkeiten im Rechenwerk

11.3.1. Vergleich Register A mit Register H oder 0

Beim Aufbau einer Tabelle kann man dem letzten Wort einen Wert geben, der in den anderen Wörtern der Tabelle nicht vorkommt. Und dieser Wert wird als Abbruchkriterium gewertet. Damit kann die Tabelle eine variable Länge haben.

```

:
ANF= ZX 0 X1,
      MA SUM1,
      E B X1,
      SIO END+1,
      MA SUM2,
      E A X1,
      MA SUM,
      E C X1,
      HXP 2 X1,
END= S ANF,

```

$\langle \text{SUM1} + a \rangle + \langle \text{SUM2} + a \rangle = \langle \text{SUM} + a \rangle$
 $a = \langle X1 \rangle = 0, 2, 4, \dots$

Bild 11.10 Beispiel Abbruch durch Vergleich

Im Beispiel Bild 11.10. gibt es die drei Wertgruppen SUM1, SUM2 und SUM. Es soll jeweils ein Wert aus der Gruppe SUM1 zu einem Wert aus der Gruppe SUM2 addiert und das Ergebnis nach der Gruppe SUM gebracht werden. Die Wertgruppe SUM1 wird mit einem Wort abgeschlossen, das den Wert 0 hat. Wird dieses Wort mit dem Befehl E B X1 ins Register A gebracht, so wird auf Grund des Befehls SIO die Schleife verlassen. Das letzte Wort wird also nicht mehr verarbeitet.

Auf die gleiche Weise läßt sich auch ein Wort aus einer Tabelle herausuchen. Hierfür stehen jedoch spezielle Befehle zur Verfügung, die Tabellensuchbefehle. Sie ermöglichen das Durchsuchen einer Tabelle auf ein bestimmtes Wort auf einfache Weise in kürzester Zeit.

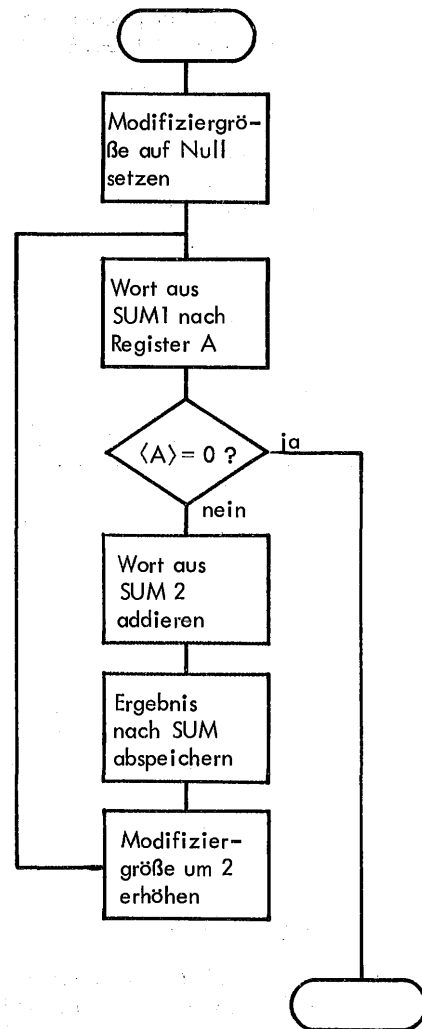
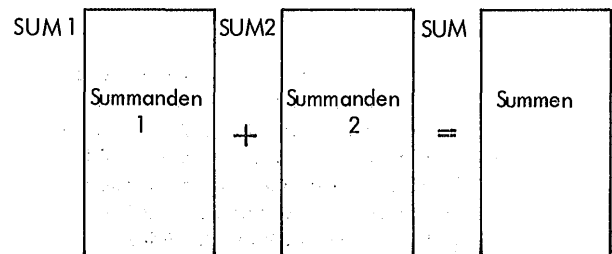


Bild 11.11 Flußdiagramm zum Beispiel Bild 11.10

11.3.2. Abfrage eines Bits

Mit den Sprungbefehlen SR und SRN kann das rechte Bit (Bit 48) im Register A abgefragt werden. Ist das Bit auf 0 gesetzt, so wird beim Befehl SRN gesprungen, ist es auf 1 gesetzt, so wird beim Befehl SR gesprungen. Soll ein beliebiges Bit im Register A, Q, D oder H abgefragt werden, so kann dies mit dem Befehl SBIT geschehen. Es wird gesprungen, wenn das im Adressenteil angegebene Bit auf 1 gesetzt ist. Der Befehl SBIT ist ein relativer Sprungbefehl, d.h. im Adressenteil steht, um wieviel Befehle vor oder zurückgesprungen werden soll.

11.3.3. Abfrage der Typenkennung

Die Typenkennung kann als Abbruchkriterium für Schleifen verwendet werden, indem das letzte Wort einer Gruppe von Wörtern auf eine andere Typenkennung gesetzt wird. Bei den Tabellensuchbefehlen wird die Typenkennung ebenfalls als Abbruchkriterium verwendet, das Tabellenende wird durch ein Wort anderer Typenkennung beendet.

Im Bild 11.11. wurde das Beispiel aus Bild 11.10. auf das Abbruchkriterium Typenkennung geändert.

Es ist lediglich der Befehl ST END-SP+1 A3 anders. Es ist ein relativer Sprungbefehl. Er könnte im Beispiel auch lauten: ST 7 A3. Im Flußdiagramm Bild 11.11. ist die Abfrage $\langle A \rangle = 0$ durch $\langle A \rangle = 3$ zu ersetzen.

```

:
:
ANF= ZX 0 X1,
      MA SUM1.,
      E B X1,
SP= ST END-SP+1 A3,
      MA SUM2,
      E A X1,
      MA SUM,
      E C X1,
      HXP 2 X1,
END= S ANF,
:
:

```

Bild 11.12 Beispiel Abbruch durch andere Typenkennung

```

:
:
ANF= ZX 0 X1,
      LA M,
      MA SUM1,
      E B X1,
      MA SUM2,
      E A X1,
      MA SUM,
      E CMR X1,
      HXP 2 X1,
      SMN ANF,
:
:

```

Bild 11.13 Beispiel Abbruch durch Marke

11.3.4. Marke

Zahlwörter können im Speicher mit einer Marke versehen werden. Diese Marke eignet sich auch als Abbruchkriterium. Das letzte Zahlwort einer Tabelle wird markiert. Wird ein markiertes Zahlwort ins Rechenwerk gebracht, so wird bei den meisten Bringebefehlen und arithmetischen Befehlen das Markenregister M gesetzt. In der großen Befehlsliste ist dann in der Spalte "M" ein "x" gesetzt. Das Register M kann mit den Befehlen SM und SMN abgefragt werden. Nähere Einzelheiten zur Marke sind den Abschnitten Festkommaarithmetik, Gleitkommaarithmetik und Sonstige Operationen zu entnehmen.

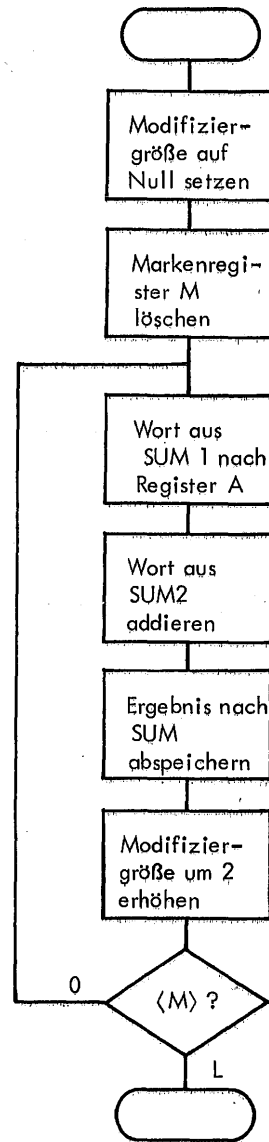


Bild 11.14 Flußdiagramm zum Bild 11.13

Das Beispiel Bild 11.13. wurde aus dem Beispiel Bild 11.10. abgeleitet. Wird die Marke als Abbruchkriterium verwendet, so soll im allgemeinen das markierte Zahlwort noch verarbeitet werden. Die Abfrage der Marke erfolgt erst am Schluß der Schleife. Das Markenregister M muß, bevor man in die Schleife kommt, gelöscht werden.



TAS - Handbuch

Dez. 68

Eine Marke kann im Speicher mit den Befehlen LMT, CMT und ZMC gesetzt und mit LMC gelöscht werden. Bei den Befehlen CMC und LC wird der Speicherinhalt verändert, die Marke bleibt jedoch erhalten. Beim Speicherbefehl CMR wird die Marke aus dem Register M mit abgespeichert.

			$\langle n \rangle_m$	$\langle n \rangle$
ZMC	n	Setze Marke im Speicher	L	erh.
CMT	n	Speichere markiert	L	$\langle A \rangle$
LMT	n	Lösche markiert	L	0
CMR	n	Speichere mit Marke aus Register	$\langle M \rangle$	$\langle A \rangle$
GMC	n	Speichere mit Marke aus Speicher	erh.	$\langle A \rangle$
LC	n	Lösche Speicher	erh.	0
LMC	n	Lösche Marke im Speicher	0	erh.

Bild 11.15 Möglichkeiten der Beeinflussung der Marke

11.3.5. Alarmer

Bedingt können auch der Typenkennungsalarm und der Bereichsüberschreitungsalarm als Abbruchkriterium verwendet werden. Bei einer Reihe von arithmetischen Befehlen tritt ein TK-Alarm auf, wenn ein Wort mit einer nicht zugelassenen Typenkennung verarbeitet wird. Dieser TK-Alarm führt nicht sofort zur Unterbrechung und kann mit dem nachfolgenden Befehl SAT abgefangen werden. Der Befehl SAT bewirkt einen Sprung, wenn ein TK-Alarm aufgetreten ist und löscht den Alarm. Es kann also mit Hilfe dieses Befehls aus einer Schleife herausgesprungen werden.

Der BÜ-Alarm tritt bei einer Reihe von Befehlen für arithmetische Operationen auf. Er führt ebenfalls nicht sofort zu einer Unterbrechung und kann durch den nachfolgenden Befehl SAA (Springe, wenn arithmetischer Alarm) abgefangen werden. Steht ein BÜ-Alarm an, so wird gesprungen und der BÜ-Alarm gelöscht. Es kann also mit Hilfe dieses Befehls aus einer Schleife herausgesprungen werden.

11.4. Merklichter

Mit dem Merklichterregister K stehen dem Programmierer 8 Binärstellen zur Verfügung, die er einzeln löschen, setzen und abfragen kann.

Sie sind geeignet, bestimmte Zustände vorübergehend festzuhalten. In Bild 11.16. sind die Befehle für Merklichter in einer Tabelle aufgeführt.

LZL	s_L	s_R	Beschreibung	s	
				s_L	s_R
LZL	s_L	s_R	Lösche und setze Merkllicht	L	0
NL	s		Negiere Merkllicht	invertiert	
SL	p	s	Springe, wenn eines der Merklichter gesetzt	-	
SLL	p	s	Springe, wenn eines der Merklichter gesetzt und lösche	0	
SLN	p	s	Springe, wenn alle Merklichter nicht gesetzt	-	
SNL	p	s	Springe, wenn alle Merklichter nicht gesetzt sonst lösche alle	0	

Bild 11.16 Befehle für Merklichter

TELEFUNKEN
COMPUTER

TAS - HANDBUCH

Sonstige Operationen

N

INHALT



SONSTIGE OPERATIONEN

<u>1.</u>	<u>SCHIFTEN</u>	1 - 1
1.1.	Spezifikationen	1 - 1
1.2.	Gleitkommazahlen	1 - 2
1.3.	Festkommazahlen, Kurzschift	1 - 3
1.4.	Festkommazahlen, Langschift	1 - 3
1.5.	Nichtzahlwörter	1 - 5
1.6.	Schift im Register B	1 - 6
1.7.	Bereichsüberschreitung	1 - 6
<u>2.</u>	<u>SPRUNGBEFEHLE</u>	2 - 1
2.1.	Art des Sprunges	2 - 2
2.2.	Keine Abhängigkeit	2 - 3
2.3.	Abhängigkeit vom Register A	2 - 4
2.4.	Abhängigkeit vom Register B	2 - 5
2.5.	Sonstige Abhängigkeit	2 - 5
<u>3.</u>	<u>SETZEN - LÖSCHEN</u>	3 - 1
3.1.	Rechenwerksregister	3 - 1
3.2.	Indezelle und Register B	3 - 2
3.3.	Speicher	3 - 3
3.4.	Typenkennung	3 - 4
3.5.	Marke	3 - 4
3.6.	Merklichter	3 - 5
3.7.	Unterprogrammregister	3 - 6
3.8.	Indexbasis	3 - 6
<u>4.</u>	<u>TABELLEN DURCHSUCHEN</u>	4 - 1
4.1.	Durchsuchen auf Identität	4 - 2
4.2.	Durchsuchen auf größer oder gleich	4 - 4
4.3.	Logarithmisch durchsuchen	4 - 4
4.4.	Durchsuchen auf Maximum oder Minimum	4 - 7
<u>5.</u>	<u>WORTGRUPPEN</u>	5 - 1
5.1.	Wortgruppentransport	5 - 1
5.2.	Register sicherstellen	5 - 2
<u>6.</u>	<u>BOOLESCHE OPERATIONEN</u>	6 - 1
6.1.	Konjunktion	6 - 1
6.2.	Disjunktion (Adjunktion)	6 - 2
6.3.	Antivalenz	6 - 2
6.4.	Negation	6 - 3
6.5.	Zusammensetzen	6 - 3

N

1. SCHIFTEN

Beim Schiften werden gemäß der Spezifikation s die Binärstellen im angegebenen Register nach rechts oder links verschoben, und zwar um soviel Stellen wie mit p angegeben sind. Die Typenkennung und auch die Dreierprobenbits werden nicht mitgeschifft.

Für das Schiften von Indexgrößen im Register B des Befehlswerks steht der Befehl SHB $s p$ zur Verfügung. Er ist im Abschnitt 1.6. näher erläutert.

Im Rechenwerk kann entweder im Register A oder im Register Q geschifft werden, außerdem auch in beiden Registern gleichzeitig, dann jedoch unabhängig voneinander; darüberhinaus können die beiden Register A und Q als doppelt langes Register zusammengefaßt und gemeinsam geschifft werden. Für den Schift im Rechenwerk steht der Befehl SH $s p$ zur Verfügung. Nähere Einzelheiten zu diesem Befehl sind in den Abschnitten 1.1. bis 1.5. beschrieben.

1.1. Spezifikationen

Der Befehl SH $s p$ gibt mit s die Art des Schifts an und mit p die Anzahl der Binärstellen, die geschifft werden sollen. p kann die Werte von ± 0 bis $+127$ annehmen. Eine Zahl, die über die Anzahl der zu schiftenden Binärstellen im Register hinausgeht, ist nicht sinnvoll. Wird für $p \pm 0$ angegeben, so erfolgt zwar eine Ausführung des Befehls, jedoch ohne einen Schift durchzuführen.

In einem Befehl können die folgenden Spezifikationen angegeben werden:

- s_1 : leer = kein Schift
- s_2 :
 - A = Register A
 - Q = Register Q
 - AQ = Register A und Q getrennt
 - Z = Register A und Q zusammen
- s_3 :
 - leer = Rechtsschift
 - L = Linksschift
- s_4 :
 - leer = gestreckter Schift
 - K = Kreisschift
- s_5 :
 - leer = ohne Rundung
 - R = mit Rundung
- s_6 :
 - leer = abhängig von der Typenkennung
 - U = unabhängig von der Typenkennung
- s_7 :
 - leer = nicht zählen
 - B = zählen der aus A geschifteten L-Bits

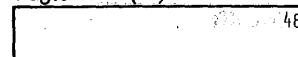
Die Spezifikationen s_1 bis s_6 können in beliebiger Reihenfolge gemeinsam angegeben werden, wobei allerdings einige Kombinationen nicht ausgeführt werden, da sie nicht sinnvoll sind.

1.1.1. Angabe des Registers (s_1)

Wird keiner der aufgeführten Buchstaben angegeben, so wirkt der Befehl wie ein Leerbefehl, d.h. es erfolgt außer der üblichen Erhöhung des Befehlsfolgeregisters keinerlei Ausführung. Bei $s_1 =$ leer und gleichzeitig $s_6 = B$ wird der Schifftzähler Y auf 0 gesetzt.

Ist als Spezifikation A bzw. Q aufgeführt, dann wird im Register A bzw. Q geschifft. Weist der Inhalt des Registers die Typenkennung 0 oder 1 auf, so wird er als Festkommazahl aufgefaßt und wie in Abschnitt 1.3. beschrieben geschifft. Hat dagegen der Registerinhalt die Typenkennung 2 oder 3, so wird er, entsprechend den in Abschnitt 1.5. dargelegten Erläuterungen als Nichtzahlwort geschifft.

Register A (Q)

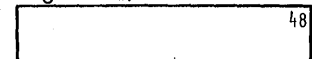


Wenn A und Q gleichzeitig angegeben sind, so werden beide Register zwar gleichzeitig, dabei jedoch völlig unabhängig voneinander geschifft.

Register A

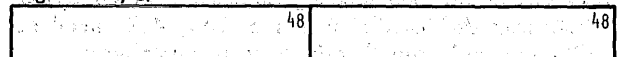


Register Q



Bei Angabe des Buchstaben Z werden die Register A und Q zu einem doppelt langen Register zusammengefaßt, wobei das Register Q die rechte Verlängerung des Registers A darstellt. Die höhere der beiden Typenkennungen bestimmt, ob der Inhalt des doppelt langen Registers als Zahlwort oder als Nichtzahlwort behandelt wird.

Register A, Q



1.1.2. Schiffrichtung (s_2)

Ist die Schiffrichtung nicht spezifiziert, wird immer nach rechts geschifft. Soll dagegen nach links geschifft werden, so ist die Angabe des Buchstaben L erforderlich.

1.1.3. Gestreckter Schiff - Kreisschiff (s_3)

Ohne eine besondere Angabe erfolgt stets ein gestreckter Schiff. Das bedeutet gleichzeitig, daß bei einem Rechtsschiff die rechts und beim Linksschiff die links herausgeschifteten Binärstellen verloren gehen. Bei Zahlwörtern werden vorzeichengleiche Stellen und bei Nichtzahlwörtern Nullen nachgezogen. Ist die Spezifikation $s_1 = Z$ (doppelt langes Register A, Q) vorhanden, dann werden beim gestreckten Schiff eines Zahlworts die Vorzeichen im rechten Register umschifft.

Für einen Kreisschiff muß der Buchstabe K angegeben werden. Dadurch wird das Ende des Registers mit dem Anfang verbunden, wobei also diejenigen Binärstellen, die man auf der einen Seite des Registers herausgeschifft hat, auf der anderen Seite des Registers wieder hineingeschifft werden und daher nicht verloren gehen. Die Vorzeichenstellen von Zahlwörtern werden bei einem Kreisschiff mitgeschifft.

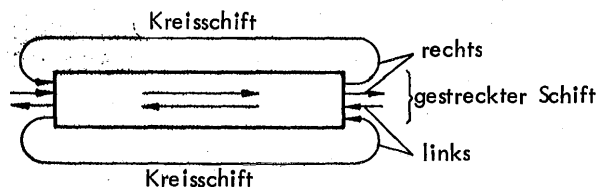


Bild 1.1. Gestreckter Schiff - Kreisschiff

1.1.4. Rundung (s_4)

Soll nach dem Schiften eine Rundung durchgeführt werden, so muß als Spezifikation der Buchstabe R angegeben werden. Eine Rundung ist nur bei Zahlwörtern und hier auch nur in einem gestreckten Schiff nach rechts (bei Spezifikation 'Z' auch nach links) sinnvoll. In allen anderen Fällen ist daher die Angabe von R bedeutungslos. Abhängig von der letzten, rechts herausgeschifteten Stelle (beim Befehl SH ZLR 0 in Abhängigkeit von $(Q)_3$) wird nach dem Schiff eine Rundung vorgenommen (im Abschnitt "Festkomma-Arithmetik" sind unter 2.9., 4.2. und 4.7. weitere Einzelheiten über die Rundung aufgeführt).

In diesem Sinne sind also nur die folgenden Kombinationen sinnvoll:

AR	ARB
QR	-
AQR	AQRB
ZR	ZRB
ZLR	ZLRB

Die beiden Spezifikationen ZLR und ZLRB bedeuten einen Langschiff nach links und bilden eine Ausnahme. Das doppelt lange Register A, Q wird nach dem Schiften auf einfache Länge gekürzt. Das Ergebnis steht gerundet im Register A, während das Register Q auf +0 gelöscht wird (siehe Abschnitte 1.4.2. und 1.4.4.).

1.1.5. Abhängigkeit von der Typenkennung

Ohne eine besondere Angabe sind alle Schiffe abhängig von der Typenkennung. Bei zwei Registern (AQ bzw. Z) bestimmt die höhere der beiden Typenkennungen die Ausführung. Zahlwörter mit Typenkennung 0 oder 1 werden nach Festkommaart geschifft (Abschnitt 1.3. und 1.4.), während Nichtzahlwörter mit Typenkennung 2 oder 3 als Bitmuster, wie in Abschnitt 1.5. beschrieben, geschifft werden.

Ist als Spezifikation der Buchstabe U angegeben, so wird der Inhalt der Register, unabhängig von der Typenkennung, immer wie bei Nichtzahlwörtern behandelt, d.h. es wird immer ein Bitmuster geschifft (Abschnitt 1.5.).

1.1.6. Zählen der besetzten Bits (s_5)

Die Angabe des Buchstaben B hat zur Folge, daß die aus dem Register A rechts oder links herausgeschifteten Bits, die auf "L" gesetzt sind, im Register Y (Schiffzähler) gezählt werden. Mit Hilfe der Befehle R und RT ist es möglich, auf den Inhalt des Registers Y zurückzugreifen.

Der Buchstabe B kann beliebig kombiniert werden. Da jedoch nur die Bits des Registers A gezählt werden, ist B bei einer Spezifikation Q nur insofern von Bedeutung, als der Inhalt des Registers Y auf Null gesetzt wird.

1.2. Gleitkommazahlen (TK = 0)

Gleitkommazahlen werden wie Festkommazahlen geschifft, d.h. Mantisse und Exponent werden nicht getrennt behandelt. Die Wirkung ist mit der in Abschnitt 1.3. und 1.4. beschriebenen identisch. Lediglich dann, wenn die Spezifikation U angegeben ist, werden Gleitkommazahlen als Bitmuster geschifft (Abschnitt 1.5.).

1.3. Festkommazahlen, Kurzschiff (TK = 1)

Die ersten Binärstellen (Bit 1 und 2) bei Festkommazahlen stellen die Vorzeichen der Zahl dar. Sie sind im allgemeinen gleich. Hat die Zahl jedoch den zulässigen Bereich überschritten, ($\geq 2^{46}$) so ist das zweite Bit verschieden vom ersten und zeigt damit an, daß der Bereich überschritten und die Zahl übergelaufen (OL...) oder untergelaufen (LO...) ist; als Vorzeichen gilt dann das 1. Bit. Diese Zahlen werden ebenfalls geschiftet, jedoch sollte das Schiften von über- oder untergelaufenen Zahlwörtern vermieden werden (siehe auch Abschnitt 1.7.).

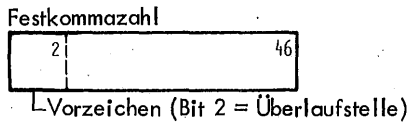


Bild 1.2. Festkommazahl einfacher Wortlänge

1.3.1. Gestreckter Rechtsschiff

Beim gestreckten Rechtsschiff eines Zahlwortes gehen die rechts herausgeschifteten Stellen verloren. Links werden vorzeichengleiche Stellen (entsprechend dem 1. Bit) nachgezogen. Ein Schift um p Stellen bedeutet eine Division durch 2^p . Vorzeichenstellen werden mitgeschiftet.

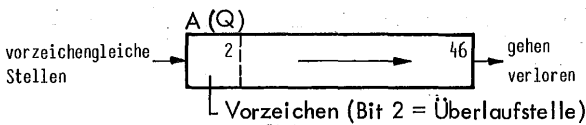


Bild 1.3. Gestreckter Rechtsschiff, kurz

Ist durch die Spezifikation R eine Rundung gewünscht, so wird die Zahl nach dem Schiften in Abhängigkeit von der zuletzt rechts herausgeschifteten Binärstelle gerundet.

Wird bei einem Schift im Register A ein über- bzw. untergelaufenes Zahlwort um eine Stelle nach rechts geschifft, dann kann bei der Spezifikation R (Runden) erneut ein über- bzw. untergelaufenes Zahlwort entstehen. In diesem Fall erfolgt BÜ-Alarm.

1.3.2. Gestreckter Linksschiff

Beim gestreckten Linksschiff werden rechts vorzeichengleiche Stellen (entsprechend dem 1. Bit) nachgezogen. Die links herausgeschifteten Stellen gehen verloren. Ein Schift um p Stellen bedeutet eine Multiplikation mit 2^p . Vorzeichenstellen werden mitgeschiftet.

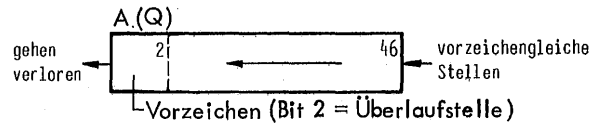


Bild 1.4. Gestreckter Linksschiff, kurz

Die Angabe einer Rundung durch die Spezifikation R ist beim gestreckten Linksschiff bedeutungslos.

Läuft während des Schiftens durch die zweite Binärstelle ein vom Vorzeichen verschiedenes Bit, so erfolgt BÜ-Alarm; der Schiftbefehl wird jedoch auch in diesem Fall bis zu Ende ausgeführt.

Über die Wirkung einer zusätzlichen Spezifikation $s_B=B$ siehe Abschnitt 1.1.6.

Bei untergelaufenen Zahlwörtern wird ein BÜ-Alarm gegeben, wenn $p \geq 5$ ist.

1.3.3. Kreisschiff

Bei Anwendung der Spezifikation K wird das Ende des Registers mit dem Anfang verbunden und die rechts bzw. links herausgeschifteten Bits links bzw. rechts wieder hineingeschiftet. Alle Binärstellen bleiben unverändert; die Vorzeichenstellen werden mitgeschiftet. Der Kreisschiff eines Zahlwortes unterscheidet sich nicht von dem Schift eines Nichtzahlwortes.

Eine Angabe der Spezifikationen R und U bleibt ohne Wirkung; es kann kein BÜ-Alarm auftreten.

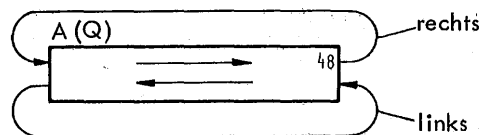


Bild 1.5. Kreisschiff rechts und links, kurz

1.4. Festkommazahlen, Langschiff (TK = 1)

Bei einem Langschiff werden die beiden Register A und Q zu einem doppelt langen Register zusammengefügt. In der Spezifikation wird der Buchstabe Z angegeben. Evtl. gleichzeitig aufgeführte Buchstaben A oder Q sind bei einem Langschiff bedeutungslos.

Die Vorzeichenstellen werden beim Rechts- und Linksschiff unterschiedlich behandelt. Beim Kreisschiff haben die Vorzeichenstellen keinen Einfluß auf die Art des Schifts.

1.4.1. Gestreckter Rechtsschift

Beim gestreckten Langschift nach rechts wird die 48. Binärstelle des Registers A mit der 3. Binärstelle des Registers Q verbunden; die Vorzeichenstellen des Registers Q werden also nicht mitgeschifft. Ist der Inhalt des Registers Q über- oder untergelaufen, so entsteht ein falsches Ergebnis.

Die Vorzeichenstellen (1. Bit) der Register A und Q müssen gleich sein. Sind sie ungleich, dann werden alle Binärstellen des Registers Q invertiert, d.h. bei der Zahl im Register Q wird das Vorzeichen gewechselt. Als Vorzeichen des doppelt langen Registers gilt dann die 1. Binärstelle des Registers A. Durch diesen Vorgang verändert sich jedoch der Wert der Festkommazahl.

Falls vorher feststeht, daß die Vorzeichen ungleich sein können, ist es deshalb zweckmäßig, vor der Ausführung des Schiftbefehls mit dem Befehl VAQ die Vorzeichen anzugleichen. Der Befehl VAQ bewirkt gleichzeitig eine Richtigestellung des Zahlenwerts (siehe auch unter "Festkomma-Arithmetik", Abschnitte 4.1. und 4.7.).

Beim Schiften werden links vorzeichengleiche Stellen (entsprechend dem 1. Bit von A) nachgezogen. Die rechts herausgeschifteten Stellen gehen verloren.

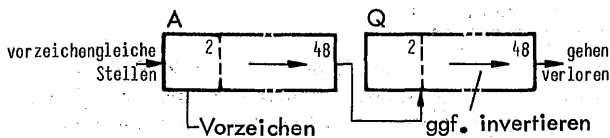


Bild 1.6. Gestreckter Rechtsschift, lang

Wird durch die Spezifikation R eine Rundung gewünscht, so wird nach dem Schiften das doppelt lange Zahlwort (94 Binärstellen) in Abhängigkeit von der zuletzt rechts herausgeschifteten Stelle gerundet.

Ist der Inhalt des Registers A über- bzw. untergelaufen und wird um eine Stelle nach rechts geschifft, so kann bei der Spezifikation R (Runden) erneut ein über- bzw. untergelaufenes Zahlwort entstehen. In diesem Fall wird BÜ-Alarm gegeben.

1.4.2. Gestreckter Linksschift

Beim gestreckten Langschift nach links (Spezifikation ZL) wird die 48. Binärstelle des Registers A mit der 3. Binärstelle des Registers Q verbunden. Die Vorzeichenstellen vom Register Q werden also nicht mitgeschifft. Ist der Inhalt des Registers Q über- bzw. untergelaufen, so entsteht ein falsches Ergebnis.

Die Vorzeichenstellen (1. Bit) der Register A und Q müssen gleich sein. Im anderen Fall werden alle Binärstellen des Registers A invertiert, d.h. bei der Zahl im Register A wird das Vorzeichen gewechselt. Als Vorzeichenstelle des doppelt langen Registers gilt dann die 1. Binärstelle des Registers Q. Dabei verändert sich der Wert der Festkommazahl. Deshalb ist es zweckmäßig, mit dem Befehl VAQ vor dem Schiften die Vorzeichen anzugleichen (siehe Abschnitt 1.4.1.).

Beim Schiften werden rechts vorzeichengleiche Stellen (1. Bit von Q) nachgezogen. Die links herausgeschifteten Stellen gehen verloren. Nach dem Schiften werden die Vorzeichenstellen des Registers A (1. und 2. Bit) und die zweite Vorzeichenstelle des Registers Q der ursprünglichen Vorzeichenstelle (1. Bit von Q) angeglichen.

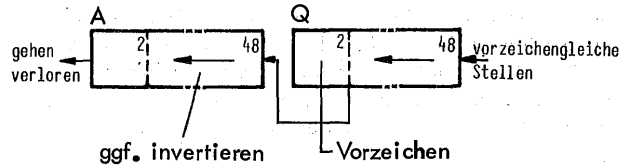


Bild 1.7. Gestreckter Linksschift, lang

Ist als Spezifikation ein R (Rundung) angegeben, so wird nach dem Schift der Inhalt des Registers A in Abhängigkeit von der 3. Binärstelle im Register Q gerundet; dabei wird das Register Q auf +0 gelöscht. Man erreicht also auf diese Weise (mit gleichzeitiger Rundung) eine Verkürzung des doppelt langen Zahlworts auf einfache Länge. Wenn nach der durchgeführten Rundung die beiden Vorzeichen im Register A ungleich sind, erfolgt BÜ-Alarm. Bei Anwendung des Befehls SH mit der Spezifikation ZLR 0 unterbleibt dagegen der BÜ-Alarm, wenn die Vorzeichen in Register A ursprünglich 0L und alle Bits in A wie auch das 3. Bit in Q L oder aber die Vorzeichen in A L0 und alle Bits in A wie auch das 3. Bit in Q 0 lauteten.

Läuft während des Schiftens durch die zweite Binärstelle des Registers A ein vom Vorzeichen (entsprechend dem 1. Bit von Q) verschiedenes Bit, so wird ein BÜ-Alarm gegeben; der Schiftbefehl wird jedoch auch in diesem Fall zu Ende geführt.

Bei untergelaufenem Zahlwort im Register A (im Fall des Invertierens von Register A nach der Invertierung) wird BÜ-Alarm gegeben, wenn $p \geq 5$ ist.

1.4.3. Kreisschift

Wird die Spezifikation K (bzw. LK) verwendet, so wird das Ende des Registers A mit dem Anfang des Registers Q verbunden und das Ende des Registers Q mit dem Anfang des Registers A (bzw. umgekehrt). Die rechts bzw. links (Spezifikation LK) herausgeschifteten Bits werden beim anderen Register links bzw. rechts wieder hineingeschifft; die Vorzeichenstellen werden mitgeschifft. Der Kreisschift eines Zahlworts unterscheidet sich nicht von dem Schift eines Nichtzahlworts.

Die Spezifikationen R und U sind beim Kreisschift ohne Bedeutung; ein BÜ-Alarm kann nicht auftreten.

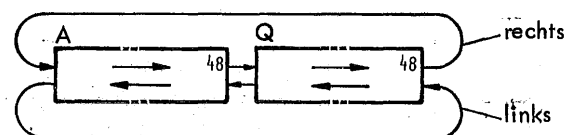


Bild 1.8. Kreisschift rechts und links, lang

1.4.4. Verkürzung

Durch Schiften kann eine doppelt lange Festkommazahl in eine Zahl einfacher Wortlänge umgewandelt werden. Dabei entfällt entweder die rechte oder die linke Hälfte der ursprünglich doppelt langen Zahl.

Vor jeder Verkürzung müssen mit Hilfe des Befehls VAQ die Vorzeichen angeglichen werden.

Eine Verkürzung um die rechte Hälfte erreicht man durch Rundung mit Hilfe des Befehls SH ZLR 0 (oder dadurch, daß man auf die doppelt lange Zahl Befehle anwendet, die nur das Register A als Operand haben).

Soll die doppelt lange Zahl um die linke Hälfte verkürzt werden, so schifft man die Zahl mit dem Befehl SH ZL 46. Bei Angabe der Spezifikation B werden gleichzeitig die evtl. herausgeschifteten L-Bits im Schifftzähler gezählt (siehe auch Abschnitt 1.4.2.).

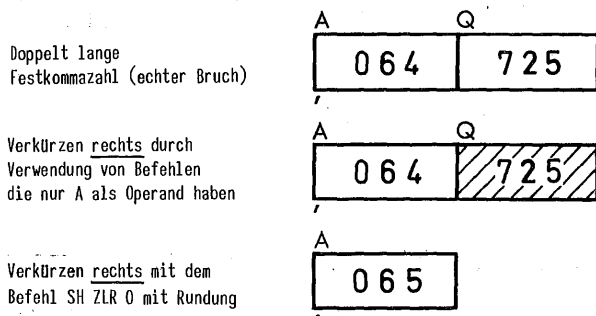


Bild 1.9. Verkürzen einer doppelt langen Festkommazahl, rechts

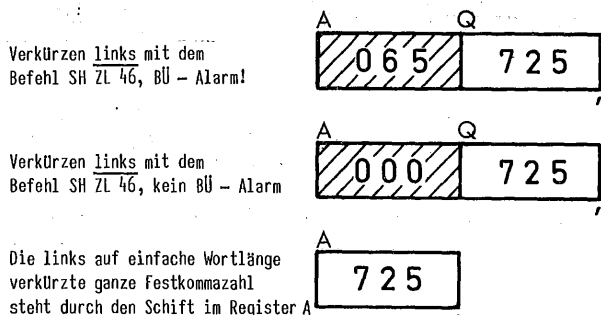


Bild 1.10 Verkürzen einer doppelt langen Festkommazahl, links

1.5. Nichtzahlwörter

Hat eines der im Befehl aufgeführten Register die Typenkenennung 2 oder 3, dann wird immer ein Bitmuster geschifft. Ist ein Schift unabhängig von der Typenkenennung erwünscht, so erfolgt durch die Angabe der Spezifikation U bei allen Typenkennungen ein Schift des Bitmusters. Vorzeichen haben auf diese Schiftart keinen Einfluß und werden immer mitgeschifft. Die Spezifikation Rundung ist bei Typenkennung 2 oder 3, bzw. bei gleichzeitiger Spezifikation U bedeutungslos. Ein BÜ-Alarm tritt nicht auf.

1.5.1. Gestreckter Kurzschift

Die links, bzw. rechts herausgeschifteten Bits gehen verloren. Es werden stets Nullen nachgezogen.

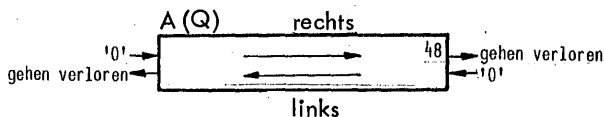


Bild 1.11. Gestreckter Kurzschift bei Nichtzahlwörtern

1.5.2. Gestreckter Langschift

Bei der Spezifikation Z wird das Ende des Registers A mit dem Anfang des Registers Q verbunden; dabei gehen die links bzw. rechts herausgeschifteten Bits verloren. Es werden stets Nullen nachgezogen.

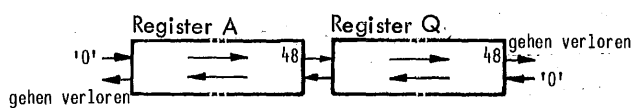


Bild 1.12. Gestreckter Langschift bei Nichtzahlwörtern

1.5.3. Kurzschift im Kreis

Bei der Spezifikation K (bzw. LK) wird das Ende des Registers (A oder Q) mit seinem Anfang verbunden (bzw. umgekehrt). Die rechts bzw. links (Spezifikation LK) herausgeschifteten Bits werden links bzw. rechts wieder hineingeschifft.

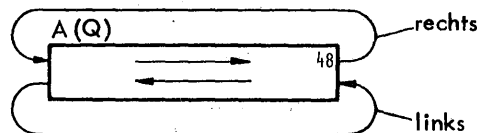


Bild 1.13. Kurzschift im Kreis bei Nichtzahlwörtern

1.5.4. Langschift im Kreis

Bei der Spezifikation ZK (bzw. ZLK) wird das Ende des einen Registers mit dem Anfang des anderen Registers verbunden (bzw. umgekehrt). Die rechts bzw. links (Spezifikation ZLK) herausgeschifteten Bits werden in das andere Register links bzw. rechts wieder hineingeschifft.

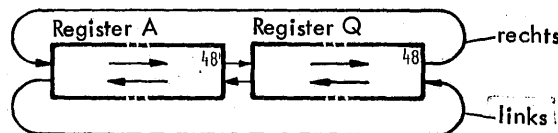


Bild 1.14. Langschift im Kreis bei Nichtzahlwörtern

1.6. Schift im Register B

Mit dem Schiftbefehl SHB s p kann im Register des Befehlswerks nach rechts (Spezifikation s = R) oder nach links (Spezifikation s = L) geschiftet werden. Die Zahl p gibt die Anzahl der Schiftschritte an und kann den Wert 0 bis 255 annehmen. Neben p muß stets eine Spezifikation angegeben werden.

Intern gesehen erfolgt ein Schift nach rechts, wenn das 9. Bit des Befehlswortes 0, nach links, wenn dieses Bit 1 lautet.

Trotz der möglichen 0 ... 255 Schiftschritte werden nur maximal 24 Schritte ausgeführt, da damit der Inhalt des Registers B bereits 0 ist.

Beim Rechtsschift gehen die rechts herausgeschifteten Bits verloren; links werden Nullen nachgezogen. Dagegen gehen beim Linksschift die links herausgeschifteten Bits verloren, während rechts Nullen nachgezogen werden.

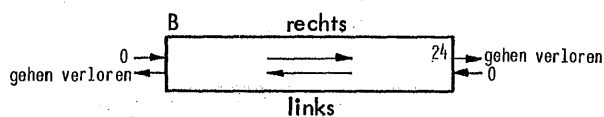


Bild 1.15. Schift im Register B

Das Vorzeichen hat keinen Einfluß auf die Schiftart. Ein BÜ-Alarm tritt nicht auf.

1.7. Bereichsüberschreitung

Eine Bereichsüberschreitung kann nur bei Zahlwörtern vorkommen. Im Normalfall sollten Über- oder untergelaufene Zahlwörter nicht geschiftet werden. Im Abschnitt 1.7.1. ist der Normalfall beschrieben.

War dagegen das Zahlwort bereits vor dem Schiften Über- oder untergelaufen, so ergeben sich Sonderfälle, die zwar nur selten auftreten, bei denen aber bei einem auftretenden Fehler nicht immer ein BÜ-Alarm gegeben wird. Diese Sonderfälle sind im Abschnitt 1.7.2. und 1.7.3. näher beschrieben.

1.7.1. Normalfall

Ist das Zahlwort nicht über- oder untergelaufen, dann kann in zwei Fällen ein BÜ-Alarm auftreten:

Läuft während des Linksschifts ein Bit, das nicht den Vorzeichenstellen (Bit 1,2) gleich ist, in die 2. Binärstelle des Registers (beim Langschift nur im Register A), so wird BÜ-Alarm gegeben. Es muß also mit einem falschen Ergebnis gerechnet werden. Nur dann, wenn dieser Vorgang beim letzten Schiftschritt erfolgt, ist das Ergebnis zwar richtig, aber über- bzw. untergelaufen.

Wird bei einem gestreckten Langschift nach links mit Rundung um 0 Stellen geschiftet (Befehl SH ZLR 0), so kann ein BÜ-Alarm auftreten, wenn im Register A der Wert $\pm 2^{48} - 1$ steht und wenn auf Grund der Rundung in Abhängigkeit von der 3. Binärstelle im Register Q aufgerundet wird. Das Ergebnis ist dann zwar richtig, aber über- bzw. untergelaufen.

1.7.2. Gestreckter Linksschift

War das Zahlwort bereits vor dem Schiften Über- bzw. untergelaufen, so ist das Ergebnis falsch, wenn um mindestens 1 Stelle geschiftet wird. Ein BÜ-Alarm erfolgt nur dann, wenn durch das Schiften in die zweite Binärstelle des Registers (beim Langschift nur im Register A) ein Bit hineinläuft, das von den Vorzeichenstellen des Zahlworts verschieden ist oder wenn ein untergelaufenes Zahlwort mindestens um 5 Stellen nach links geschiftet wird.

1.7.3. Gestreckter Rechtsschift mit Rundung

Hat das Zahlwort im Register A (und zwar nur in A) den Wert $\pm 2^{47} - 1$, d.h. also den maximalen Über- oder untergelaufenen Wert (0 L...L oder L 0...0), so entsteht bei einem Schift um eine Stelle nach rechts und anschließender Rundung wieder ein Über- oder untergelaufenes Wort. Es wird BÜ-Alarm gegeben. Das Ergebnis ist richtig.

Tritt der vorstehende Fall bei einem gestreckten Langschift nach links ein, und wird mit dem Befehl SH ZLR 0 um 0 Stellen (mit gleichzeitiger Rundung) geschiftet, so erfolgt eine Rundung in Abhängigkeit vom 3. Bit im Register Q. Wird dabei aufgerundet, so entsteht ein falsches Ergebnis, ohne daß ein BÜ-Alarm gegeben wird.

2. SPRUNGBEFEHLE

Sprungbefehle dienen zur Programmverzweigung. Zu diesem Zweck kann mit Hilfe der Sprungbefehle

- der Inhalt des Registers A
- der Inhalt des Registers B
- die Markenstelle bei Zahlwörtern
- die Typenkennung
- die Merklichter (im Register K)
- Alarmmeldungen
- der Exponent bei Gleitkommazahlen

abgefragt werden. Ist die beim Befehl angegebene Bedingung erfüllt, so wird das Befehlsfolgeregister (Register F) auf einen neuen Wert gesetzt und an der Stelle im Programm fortgefahren, die das Register F anzeigt. Beim Befehl ist jeweils angegeben, auf welchen Wert das Register F bei erfüllter Sprungbedingung gesetzt wird.

Ist die Bedingung nicht erfüllt, so wird der Inhalt des Registers F automatisch um 1 erhöht und der auf den Sprungbefehl folgende Befehl wird ausgeführt. Die Erhöhung um 1 wird bei allen Befehlen vorgenommen und ist bei der Beschreibung der einzelnen Befehle nicht aufgeführt.

Im Bild 2.1 sind einige Beispiele für Verzweigungen, wie sie vom Problem her gesehen werden. Im ersten Beispiel besteht die Verzweigung darin, ein bestimmtes Programmstück zu überspringen. Im zweiten Beispiel wird eine von zwei Möglichkeiten ausgewählt. In den weiteren Beispielen wird zwischen mehreren Möglichkeiten gewählt.

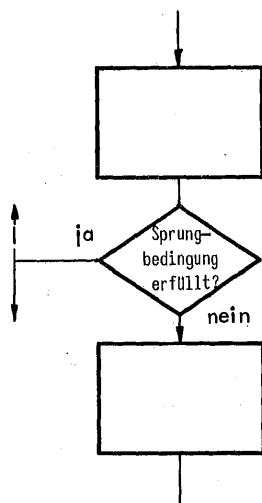


Bild 2.2 Prinzip des Sprungbefehls

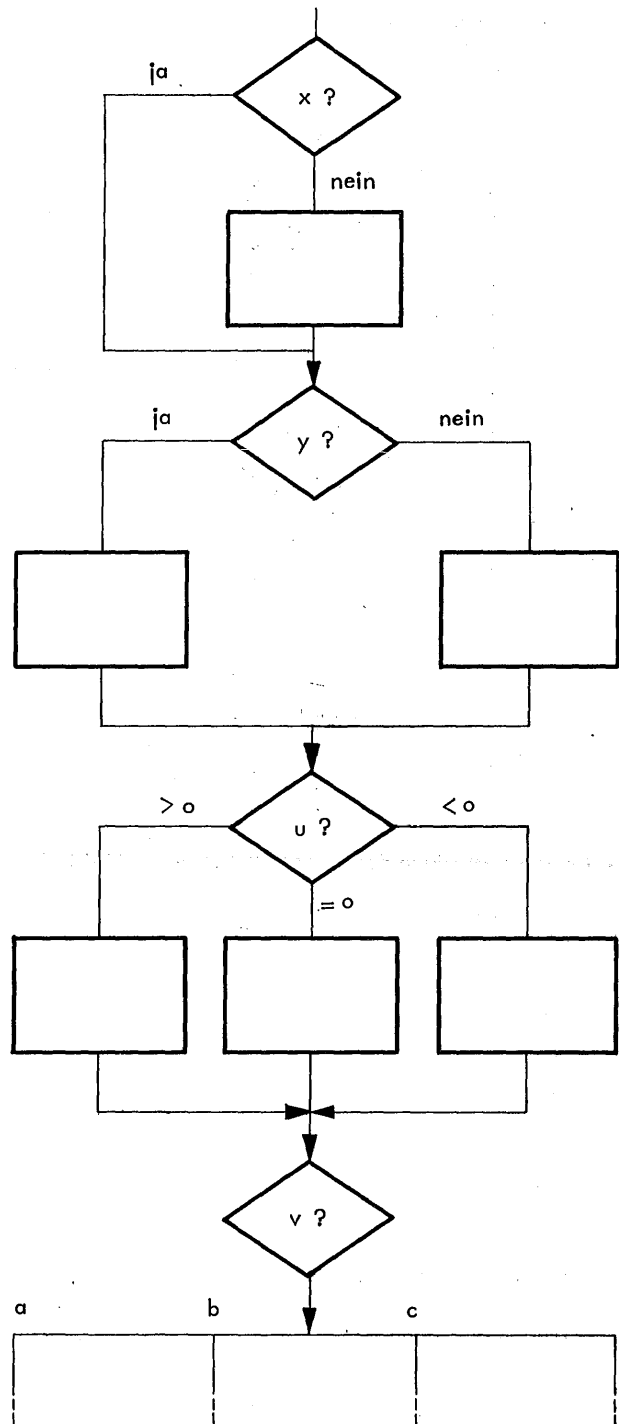


Bild 2.1 Beispiele für Verzweigungen

Befehle stehen stets hintereinander im Speicher und werden nacheinander abgearbeitet. Mit Hilfe der Sprungbefehle kann nun eine Entscheidung getroffen werden, ob fortgefahren (Sprungbedingung ist nicht erfüllt) oder abgebrochen und an einer anderen Stelle fortgefahren werden soll (Sprungbedingung ist erfüllt). Im Bild 2.2 ist dies im Prinzip gezeigt.

Durch die Verwendung mehrerer Sprungbefehle lassen sich alle möglichen Verzweigungen ausführen. Im Bild 2.3 ist gezeigt, wie das zweite Beispiel aus Bild 2.1 durch zwei Sprungbefehle gelöst wird. Besteht an einem Verzweigungspunkt eine große Zahl von Möglichkeiten, so kann es zweckmäßig sein, mit Hilfe von Tabellensuchbefehlen die Verzweigung vorzunehmen.

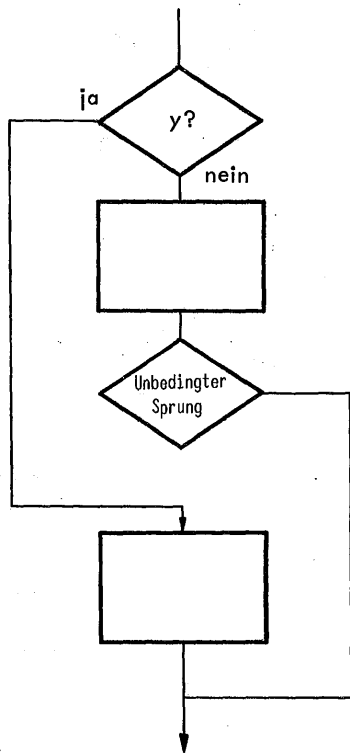


Bild 2.3 Abwandlung des zweiten Beispiels aus Bild 2.1

Sobald ein Operator mehr als 65 536 Befehle umfaßt, also mehr als eine Großseite, sind Sprungbefehle erforderlich, die in eine andere Großseite springen. Dies ist im Abschnitt 2.1.3. näher erläutert.

2.1.1. Bedingt - unbedingt

Bedingte Sprungbefehle bewirken nur dann einen Abbruch des Programmablaufs und das Fortsetzen des Programmablaufs an einer anderen Stelle, wenn eine bestimmte Bedingung erfüllt ist. Diese Art der Bedingung hängt ab von der Wirkung des Befehls.

Ist die geforderte Bedingung nicht erfüllt, so wird der sequentielle Ablauf des Programms fortgesetzt. Dabei können bei einigen Befehlen noch spezielle Wirkungen auftreten, die nicht mit der Verzweigung zusammenhängen und ggf. ebenfalls von der Sprungbedingung abhängig sind.

Bei einem unbedingten Sprung ist keine Bedingung gegeben. Es wird immer gesprungen.

Der unbedingte Sprung wird z.B. verwendet, um die im Bild 2.3 skizzierte Auswahl zwischen zwei Möglichkeiten zu realisieren, um in ein Unterprogramm zu springen, usw.

2.1. Art des Sprunges

Von der Sprungbedingung her unterscheiden wir zwischen

- bedingtem Sprung und
- unbedingtem Sprung.

In bezug auf das Sprungziel unterscheiden wir zwischen

- absoluten Sprüngen,
- relativen Sprüngen und
- indirekten Sprüngen.

Die Sprache TAS erlaubt es, bei Sprungbefehlen mit absoluter Adresse relativ zu adressieren und bei Sprungbefehlen mit relativer Adresse absolut zu adressieren. Bei den indirekten Sprüngen steht das absolute Sprungziel in einem Halbwort, das durch den Sprungbefehl adressiert wird.

2.1.2. Absolut - relativ - indirekt

Sprungbefehle, die bei der Beschreibung im Adressenteil den Buchstaben m stehen haben, sind immer absolute Sprungbefehle. Sie geben im Adressenteil stets die Adresse an, auf die gesprungen wird. Eine Besonderheit gilt dabei für die Sprungbefehle mit indirekter Adresse (SE, SFBE und SUE), die weiter unten behandelt werden.

Die Sprungbefehle, bei denen in der Beschreibung im Adressenteil der Buchstabe p steht, sind relative Sprungbefehle. Die dafür angegebene Zahl gibt an, um wieviel Befehle vor- bzw., wenn sie negativ ist, um wieviele Befehle zurückgesprungen werden soll. Das Sprungziel ist also relativ zum Befehl selbst.

Die absolute Adressierung bei Sprungbefehlen hat den Vorteil, daß der Abstand zum Sprungziel nicht ausgerechnet zu werden braucht und daß man nachträglich noch Befehle zwischen Sprungbefehl und Sprungziel einschieben oder entfernen kann.

Die relative Adressierung hat den Vorteil, daß Namen eingespart werden können, wenn der Abstand zwischen Sprungbefehl und Sprungziel überschaubar ist.

Es besteht die Möglichkeit, beide Adressierungsarten bei den relativen und bei den absoluten Sprüngen anzuwenden. Bei den absoluten Sprungbefehlen kann eine relative Adresse angegeben werden, wenn der Buchstabe R angeschlossen wird. Dies bewirkt, daß zu der Zahl, die vor dem Buchstaben R steht, die Adresse des Sprungbefehls addiert wird. Diese Summe ist dann das Sprungziel.

Bei den relativen Sprungbefehlen kann an Stelle der relativen Adresse eine symbolische Adresse (Kernspeicherbezug; siehe Abschnitt "Befehle und Adressierung") angegeben werden. Es wird vom Assembler die Differenz gebildet zwischen dem Sprungziel (symbolische Adresse) und der Adresse des Sprungbefehls. Dies ergibt die relative Adresse. Sie muß im vorgeschriebenen Bereich liegen. Im Bild 2.4 sind einige Beispiele für absolute Sprungbefehle und in Bild 2.5 für relative Sprungbefehle aufgeführt.

```

SGG +15R,
SK 3R,
SIO -10R,
S 10R,

```

Bild 2.4 Beispiele für relatives Sprungziel bei absoluten Sprungbefehlen

```

SZX ANF X1,
SL ENDE-5 8,
ST SCH+1 A3,
SZX B+A-C XB,

```

Bild 2.5 Beispiele für absolutes Sprungziel bei relativen Sprungbefehlen

Bei den Sprungbefehlen mit indirekter Adressierung steht das Sprungziel nicht im Adressenteil des Befehls. Im Adressenteil steht die Adresse eines Halbwortes, in dem dieses Sprungziel steht. Die Angabe eines relativen Sprungziels ist nicht möglich. Dies trifft für die Befehle SE, SFBE und SUE. Der Buchstabe E deutet an, daß erst nach der Ersetzung der Adresse durch den Inhalt des adressierten Halbwortes der Sprung ausgeführt wird. Diese Befehle werden benötigt, wenn ein Operator mehr als 65 536 Befehle umfaßt, um in eine andere Großseite zu springen (siehe Abschnitt 2.1.3.).

```

SPR= ANTON/A,
      SE SPR,
      SUE (ADD/A),
      SFBE (UNTPR/A),

```

Bild 2.6 Beispiele für indirekte Sprungbefehle

2.1.3. Sprung in andere Großseite

Umfaßt ein Operator mehr als 65 536 Befehle, so reicht für die Befehle eine Großseite (= 65 536 Halbörter) nicht aus. Die Befehle des Operators müssen dann auf zwei Großseiten verteilt werden.

Die meisten Sprungbefehle springen nur innerhalb einer Großseite, d.h. sie verändern nur die rechten 16 Bits des Registers F, die linken 8 Bits bleiben unverändert.

Bei den Befehlen SE, SUE und SFBE werden jedoch alle 24 Bits des Registers F neu gesetzt. Damit kann in eine andere Großseite gesprungen werden. Das gleiche gilt, wenn ein Sprungbefehl als Zweitbefehl beim Befehl MABI oder MU verwendet wird.

2.2. Keine Abhängigkeit

Nachstehend sind die unbedingten Sprungbefehle, sowie einige Befehle beschrieben, die nicht zu den Sprungbefehlen gehören, aber doch eine ähnliche Wirkung haben.

Alle unbedingten Sprungbefehle haben ein absolutes Sprungziel oder ein indirektes Sprungziel.

2.2.1. Sprung

Für den einfachen Sprung stehen die Befehle

S	m	Sprunge	$\langle F \rangle_{8-24} := m$
SE	m	Sprunge nach Ersetzung	$\langle F \rangle := \langle m \rangle$

zur Verfügung. Beim Befehl S wird im Adressenteil das Sprungziel angegeben. Beim Befehl SE steht das Sprungziel im Halbwort, das durch die Adresse adressiert wird.

Ein Modifikator zweiter Art wird in beiden Fällen zum Sprungziel addiert.

Die Befehle SFB und SFBE werden hauptsächlich zum Sprung in ein Unterprogramm verwendet und sind im Abschnitt 2.2.2. beschrieben.

N

2.3.2. Festkommazahl

Der Vergleich geht nach den angegebenen Regeln vor sich, dabei sind +0 und -0 identisch.

2.3.3. Nichtzahlwörter

Wörter mit der Typenkennung 2 oder 3 werden als Bitmuster verglichen, wobei die links stehenden Bits die höhere Wertigkeit haben, als die rechts stehenden. Ein Nichtzahlwort kann nie kleiner als Null sein. Es ist nur dann Null, wenn alle Bits Null sind. Zwei Nichtzahlwörter sind nur dann identisch, wenn sie in allen Binärstellen übereinstimmen.

2.3.4. Binärstellen

Mit den Befehlen

SR	m	Springe wenn <u>rechtes</u> Bit in A gesetzt	$\langle A \rangle_{48} = L$
SRN	m	Springe wenn <u>rechtes</u> Bit in A <u>nicht</u> gesetzt	$\langle A \rangle_{48} = 0$

kann das rechte Bit im Register A abgefragt werden. Darüber hinaus kann mit dem Befehl

SBIT	p s	Springe wenn <u>Bit</u> gesetzt	$\langle s_2 \rangle_{s_1} = L$
------	-----	---------------------------------	---------------------------------

jedes Bit im Register A, Q, D oder H abgefragt werden, ob es auf L gesetzt ist.

2.4. Abhängigkeit vom Register B

Der Inhalt des Registers B kann mit dem Wert Null verglichen werden. Dafür stehen die Befehle

SXI	m	Springe wenn Index <u>identisch</u> 0	$\langle B \rangle = \pm 0$
SXN	m	Springe wenn Index <u>nicht</u> identisch 0	$\langle B \rangle \neq \pm 0$
SXGG	m	Springe wenn Index <u>größer</u> gleich 0	$\langle B \rangle \geq \pm 0$
SXG	m	Springe wenn Index <u>größer</u> 0	$\langle B \rangle > \pm 0$
SXKG	m	Springe wenn Index <u>kleiner</u> gleich 0	$\langle B \rangle \leq \pm 0$
SXK	m	Springe wenn Index <u>kleiner</u> 0	$\langle B \rangle < \pm 0$

zur Verfügung. Das linke Bit des Registers B ist Vorzeichen. Plus und minus Null sind identisch.

Im Register B kann ebenfalls das rechte Bit abgefragt werden. Hierzu stehen die Befehle

SXR	m	Springe wenn Indexgröße <u>rechtes</u> Bit = L	$\langle B \rangle_{24} = L$
SXRN	m	Springe wenn Indexgröße <u>rechtes</u> Bit <u>nicht</u> L	$\langle B \rangle_{24} = 0$

zur Verfügung.

2.5. Sonstige Abhängigkeit

2.5.1. Indexzelle

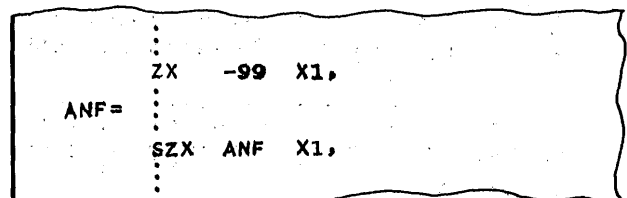
Mit dem Befehl

SZX	p i	Springe und <u>zähle</u> wenn Index <u>kleiner</u> 0	$\langle i \rangle < \pm 0$
-----	-----	--	-----------------------------

p: $\pm 0 \dots \pm 127$

kann der Inhalt einer Indexzelle mit dem Wert Null verglichen werden. Ist der Inhalt der Indexzelle kleiner als Null, so ist die Sprungbedingung erfüllt. In diesem Fall wird außerdem der Inhalt der Indexzelle um 1 erhöht.

Im Beispiel Bild 2.8 ist gezeigt, wie sich mit Hilfe des Befehls einfach Schleifen aufbauen lassen. Dazu wird die Indexzelle auf einen negativen Wert gesetzt. Er gibt an, wie oft die Schleife durchlaufen werden soll. Ist sie z.B. auf -99 gesetzt, so wird die Schleife hundertmal durchlaufen. Am Ende der Schleife wird mit dem Befehl SZX abgefragt, ob bereits der Wert Null erreicht ist, wenn nicht, wird um 1 erhöht.



Die Schleife wird 100 mal durchlaufen

Bild 2.8 Beispiel zum Befehl SZX

N

TAS - Handbuch

Juli 69

2.5.2. Binärstellen

Mit dem Befehl

SBIT	p	s	Springe wenn <u>Bit</u> gesetzt	$\langle s_2 \rangle_{s_1} = L$
------	---	---	---------------------------------	---------------------------------

s_1 : Bitnummer 1 bis 48
 s_2 : Register A, Q, D oder H

: Sprungweite $\pm 0 \dots \pm 127$

kann jede Bitstelle in den Rechenwerksregistern A, Q, D oder H abgefragt werden, ob sie auf L gesetzt ist oder nicht.

2.5.3. Marke

Zahlwörter können im Speicher markiert sein. Wird ein markiertes Zahlwort in den Speicher gebracht, so wird bei den meisten Befehlen das Markenregister M gesetzt. Bei einem nicht markierten Zahlwort wird das Markenregister M nicht verändert.

Das Markenregister kann damit abgefragt werden, ob innerhalb eines Programmbereiches ein markiertes Zahlwort geholt wurde. Zum Anfang dieses Programmbereiches muß das Markenregister Null sein und ist ggf. mit dem Befehl ZTR M zu löschen.

Zur Abfrage stehen die Befehle

SM	m	Springe wenn <u>Marke</u>	$\langle M \rangle = L$
SMN	m	Springe wenn <u>Marke nicht</u>	$\langle M \rangle = 0$

zur Verfügung. Die beiden Befehle löschen außerdem das Markenregister, so daß es für den nächsten Abfragebereich bereits auf Null gesetzt ist.

Weitere Einzelheiten zum Arbeiten mit der Marke sind dem Abschnitt "Festkommaarithmetik", 2.4., zu entnehmen.

2.5.4. Typenkennung

Die Typenkennung eines Wortes kann in den Registern A, Q, D oder H abgefragt werden. Dazu stehen die Befehle

ST	p	s	Springe wenn <u>Typenkennung</u>	$\langle s_2 \rangle_t = s_1$
STN	p	s	Springe wenn <u>Typenkennung nicht</u>	$\langle s_2 \rangle_t \neq s_1$

s_1 : Typenkennung 0, 1, 2 oder 3
 s_2 : Register A, Q, D oder H

: Sprungweite $\pm 0 \dots \pm 127$

zur Verfügung. Die Typenkennung kann zur Kennzeichnung von Wörtern dienen oder zur Abgrenzung einer Wortgruppe. Dabei ist jedoch zu beachten, daß bei einer Reihe von Befehlen die Typenkennung die Befehlsausführung beeinflusst.

2.5.5. Merklischer

Dem Programmierer stehen 8 Merklischer zur Verfügung, die mit dem Befehl LZL gesetzt und gelöscht und mit dem Befehl NL negiert werden können. Sie sind von 1 bis 8 nummeriert und können die Werte 0 und L annehmen. Mit den Befehlen

SL	p	s	Springe wenn <u>Merklischer</u>	eines der $\langle K \rangle_s = L$
SLL	p	s	Springe wenn <u>Merklischer und lösche</u>	eines der $\langle K \rangle_s = L$

s: Nummer des Merklisches
 1, 2, 3, ..., 7 und 8

: Sprungweite $\pm 0 \dots \pm 127$

wird abgefragt, ob eines der angegebenen Merklischer L ist. Ist dies der Fall, so ist die Sprungbedingung erfüllt. Beim Befehl SLL werden die angegebenen Merklischer dann gelöscht. Mit den Befehlen

SLN	p	s	Springe wenn <u>Merklischer nicht</u>	alle $\langle K \rangle_s = 0$
SNL	p	s	Springe wenn <u>Merklischer nicht sonst lösche</u>	alle $\langle K \rangle_s = 0$

s: Nummer des Merklisches
 1, 2, 3, ..., 7 und 8

: Sprungweite $\pm 0 \dots \pm 127$

wird abgefragt, ob alle angegebenen Merklischer gelöscht sind (0 sind). Ist dies der Fall, so ist die Sprungbedingung erfüllt. Beim Befehl SNL werden im anderen Fall die angegebenen Merklischer gelöscht.

Die Merklischer sind besonders geeignet, um binäre Entscheidungen (Ja-Nein-Entscheidungen) für eine Zeit zu speichern.

2.5.6. Alarmmeldung

Wird bei der Ausführung eines Befehls ein vom Befehl festgelegter Bereich überschritten, so wird ein BÜ-Alarm gegeben. Wird eine vom Befehl vorgeschriebene Typenkennung nicht eingehalten, so gibt es einen TK-Alarm. In beiden Fällen bedeutet dies das Setzen eines Bits, das diesen Alarm repräsentiert.

TK-Alarm und BÜ-Alarm führen nicht sofort zur Unterbrechung des Programms. Es können noch alle Befehle ablaufen, die nicht das Rechenwerk belegen.

Es gibt nun die Möglichkeit, diese Alarme mit den Befehlen

SAA	m	Springe wenn arithmetischer Alarm (BÜ-Alarm)	BÜ-Alarm
SAT	m	Springe wenn Alarm (Typenkennung)	TK-Alarm

abzufangen. Ihre Sprungbedingung ist erfüllt, wenn der jeweilige Alarm ansteht. Ist dies der Fall, so wird gleichzeitig der Alarm gelöscht, d.h. das zugehörige Bit wird auf 0 gesetzt.

Damit ist es möglich, im eigenen Programm diesen Alarmfall zu behandeln, und der Programmablauf wird nicht abgebrochen.

2.5.7. Gleitkommaexponent

Mit dem Befehl

SEGG	p_L	p_R	Springe wenn Exponent größer gleich	$\langle A \rangle_{41-48} \geq p_R$
------	-------	-------	-------------------------------------	--------------------------------------

p_L : Sprungweite $\pm 0 \dots \pm 127$
 p_R : Vergleichsexponent
 0...127 positiv
 NO...N127 negativ

kann der Exponent einer Gleitkommazahl, die im Register A steht und auch der Exponent einer doppelt langen Gleitkommazahl, die im Register A, Q steht, abgefragt werden. Die Sprungbedingung ist erfüllt, wenn der Exponent gleich oder größer ist als die im Adressenteil angegebene Zahl. Plus und minus Null sind dabei identisch. Hat dabei das Register A nicht die Typenkennung 0, so wird ein TK-Alarm gegeben. Als Nebenwirkung wird das Register D auf den gleichen Wert gesetzt wie das Register A.

N

TAS - Handbuch

Juli 69

3. SETZEN - LÖSCHEN

Die Setz- und Löschbefehle werden hier gemeinsam beschrieben, da das Löschen eine spezielle Art des Setzens ist. Beim Löschen wird auf den Wert Null gesetzt. Damit ist es möglich, mit den Setzbefehlen auch zu löschen.

3.1. Rechenwerksregister

Alle Register des Rechenwerks können mit Hilfe der Bringebefehle auf einen bestimmten Wert gesetzt werden. Darüber hinaus stehen spezielle Befehle zur Verfügung, die das Register A auf einen Wert setzen können.

Für das Löschen der Rechenwerksregister steht ein Befehl zur Verfügung, der gleichzeitig die Typenkennung auf einen bestimmten Wert setzt. Ferner steht ein Befehl zur Verfügung, mit dem bestimmte Teile des Registers A gelöscht werden, während der Rest unverändert bleibt.

Das Setzen und Löschen des Registers B wird im Abschnitt 3.2. behandelt.

3.1.1. Register setzen

Die allgemeinste Form, ein Register auf einen bestimmten Wert zu setzen, besteht darin, mit Hilfe der Bringebefehle eine Konstante aus dem Speicher in das Register zu bringen. Diese Konstante kann auf einfache Weise als Literal im Adressenteil des Bringebefehls angegeben werden.

Diese Form erlaubt es, alle 48 Bits des Registers und die Typenkennung auf den gewünschten Wert zu setzen.

KONST=100000,	BH	KONST,
	BH	(100000),
	BD	(('FF0ABF'/3),
	B	('000,00'/2),
	B2	(ANTON/A),

Bild 3.1 Beispiele für das Setzen der Rechenwerksregister

3.1.2. Register A setzen

Soll das Register A auf einen Wert gesetzt werden, der nicht größer als 65 535 ist, so ist dies auf einfache Weise mit den Befehlen

BA	z	Bringe Adressenteil	<A> := z
BAN	z	Bringe Adressenteil negativ	<A> := -z

z: 0...65 535 (vor Modifizierung)

möglich. Die Typenkennung wird dabei auf den Wert 1 gesetzt. Soll das Register eine andere Typenkennung bekommen, so kann anschließend mit dem Befehl ZTR die Typenkennung geändert oder das Register auf die im vorstehenden Abschnitt beschriebene Art gesetzt werden.

Der Befehl BAN ist für Nichtzahlwörter nur bedingt verwendbar. Nichtzahlwörter haben kein Vorzeichen. Es wird der gesamte Registerinhalt invertiert.

Eine Erweiterung der vorstehend genannten Befehle sind die Befehle

BAR	z	Bringe Adressenteil und reserviere	<H> := <A> <A> := z
BANR	z	Bringe Adressenteil negativ und reserviere	<H> := <A> <A> := -z

z: 0...65 535 (vor Modifizierung)

Sie bewirken zusätzlich, daß der Inhalt des Registers A vorher in das Register H gebracht wird. Dort wird er also zur späteren Verwendung sichergestellt.

Der Adressenteil der Befehle kann eine 16-Bit-Größe aufnehmen. Durch Modifizierung kann sie auf 24 Bits erweitert werden. Soll das Register A auf eine 24-Bit-Größe gesetzt werden, so ist es im allgemeinen günstiger, die im Abschnitt 3.1.1. beschriebene Art zu wählen.

BA	50000,
BAN	3500,
BAR	'',''',
BA	EMIL,

Bild 3.2 Beispiele für das Setzen des Registers A

N

TAS - Handbuch

Dez. 68

3.1.3. Register löschen

Für das Löschen der Rechenwerksregister steht der Befehl

LR	s	Lösche Register	$\langle s_2 \rangle := +0$ $\langle s_2 \rangle_t := s_1$
----	---	-----------------	---

$s_1: 0, 1, 2 \text{ oder } 3 \text{ (TK)}$
 $s_2: A, D, Q \text{ und H}$

zur Verfügung. Er löscht stets auf plus Null (alle Binärstellen 0). Gleichzeitig setzt er die Typenkennung auf den angegebenen Wert.

Beim Befehl LR können mehrere Register angegeben werden. Sie erhalten alle die angegebene Typenkennung.

LR	A3,
LR	AQ0,
LR	2D,
LR	0A,

Bild 3.3 Beispiele für das Löschen der Register

3.1.4. Teillöschung Register A

Um im Register A nur bestimmte Teile zu löschen, steht der Befehl

LA	s	Lösche in A	$\langle A \rangle_s := 0$ $\langle M \rangle := 0 \text{ nur bei } s = M$
----	---	-------------	---

zur Verfügung. In Bild 3.4 ist angegeben, welche Teile des Registers gelöscht werden können. Die Spezifikationen H und T dürfen nur einzeln verwendet werden oder mit der Spezifikation M zusammen. Alle anderen (einschließlich M) können kombiniert werden. Die Spezifikation F hat gleichzeitig die Bedeutung "ohne rechte Oktade". Die Spezifikation FE löscht das ganze Register.

LA	F,
LA	23,
LA	E2M,
LA	TM,

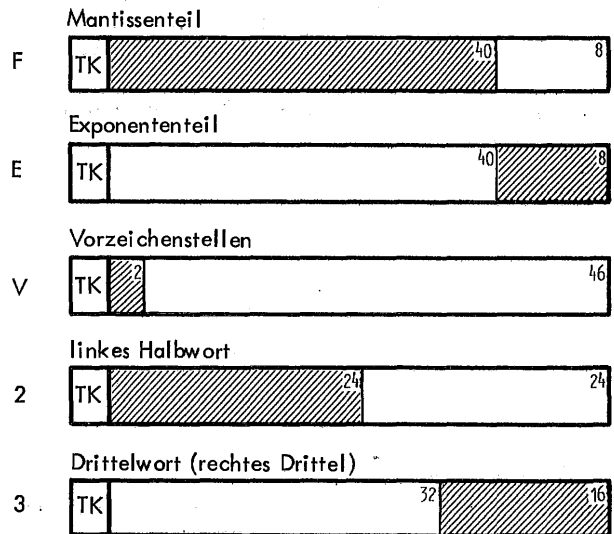
Bild 3.5 Beispiele zum Befehl LA

3.2. Indezzelle und Register B

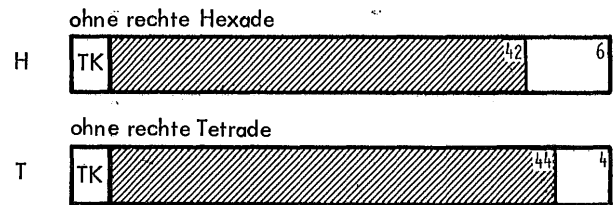
Für Adressenrechnungen ist es des öfteren erforderlich, eine Indezzelle oder das Register B auf einen definierten Wert zu setzen. In diesem Sinn kann auch das Löschen als ein Setzen auf den Wert ± 0 betrachtet werden.

Wird eine Indezzelle auf einen Wert gesetzt, so ist stets damit verbunden, daß das Register B auf den gleichen Wert gesetzt wird.

Spezifikation



können kombiniert werden



nur einzeln oder mit M erlaubt



kann mit allen kombiniert werden

Bild 3.4 Möglichkeiten beim Befehl LA

Die Möglichkeiten des Setzens richten sich nach der Größe des Wertes. Hier gibt es drei Möglichkeiten, die durch die folgenden Grenzen bestimmt sind:

- 8-Bit-Größe ± 127
- 16-Bit-Größe ± 65 535
- 24-Bit-Größe ± 8 388 607 (bzw. 16 777 215)

Dabei ist zu beachten, daß (symbolische) Adressen, soweit sie die Großseite 0 ansprechen, und die (symbolischen) Adressen der Sprungbefehle, soweit sie nur innerhalb der Großseite springen, 16-Bit-Größen sind und für die weiteren Großseiten 24-Bit-Größen.

Um eine Indezzelle auf einen Wert zu setzen, der im Bereich von -127 bis +127 liegt, gibt es den Befehl ZX.

ZX	p	i	Setze Index	$\langle i \rangle := p$ $\langle B \rangle := p$
----	---	---	-------------	--

$p: \pm 0 \dots \pm 127$

Das Register B wird auf den gleichen Wert gesetzt. Soll nur das Register B gesetzt werden, so ist dies mit einem der weiter unten aufgeführten Befehle möglich.

Sobald die Größe den Betrag von 127 überschreitet, ist das Setzen einer Indezzelle nur über zwei Befehle möglich. Es wird erst das Register B gesetzt und anschließend der Inhalt des Registers B in der Indezzelle abgespeichert. Das Abspeichern geschieht mit den Befehlen XC und XCN;

XC	i	Index:Speichern	$\langle i \rangle := \langle B \rangle$
XCN	i	Index:Speichern negativ	$\langle i \rangle := \langle B \rangle$

Um das Register B zu setzen, gibt es die Befehle

XBA	z	Index:Bringe Adressenteil	$\langle B \rangle := z$
XBAN	z	Index:Bringe Adressenteil negativ	$\langle B \rangle := -z$

$z: 0 \dots 65 535$

Die im Adressenteil des Befehls angegebene Zahl z darf maximal 65 535 sein und wird bei XBA als positive oder bei XBAN als negative Zahl ins Register B gebracht.

Werden für die Adressenrechnung 24-Bit-Größen benötigt, so können sie über die Angabe einer Konstanten als Halbwort in eine Speicherzelle gebracht werden.

Anschließend werden sie mit dem Befehl TCB in das Register B gebracht und danach mit dem Befehl XC oder XCN in eine Indezzelle.

TCB	m	Transport aus Speicher nach B	$\langle B \rangle := \langle m \rangle$
-----	---	-------------------------------	--

Etwas einfacher kann diese Konstante auch als Literal im Adressenteil des Befehls TCB angegeben werden.

TCB (131 000/H),

Der Assembler legt diese Konstante dann in einer Speicherzelle ab und setzt die Adresse dieser Zelle im Befehl TCB ein.

Soll die Indezzelle (bzw. das Register B) auf eine (Anfangs-) Adresse gesetzt werden, so kann anstelle der Zahl auch eine symbolische Adresse gesetzt werden. Liegt diese Adresse innerhalb der Großseite 0, so kann der Befehl XBA verwendet werden. Im anderen Fall wird als Konstante eine Adresskonstante verwendet.

Im Bild 3.6 sind nur Beispiele angegeben, die die Indezzelle setzen. Alle setzen gleichzeitig das Register B. Soll nur das Register B gesetzt werden, so ist (außer bei ZX) lediglich der Befehl XC bzw. XCN fortzulassen.

ZX	4	X1,
ZX	-100	X1,
ZX	0	X1,
XBA	781,	
XC	X1,	
XBAN	781,	
XC	X1,	
KONST=100000/H,		
TCB	KONST,	
XC	X1,	
TCB	(100000/H),	
XC	X1,	
TCB	(-10000/H),	
XC	X1,	
TCB	(ANF/A),	
XC	X1,	
TCB	(ANF+100/A),	
XC	X1,	

Bild 3.6 Beispiele für das Setzen einer Indezzelle

3.3. Speicher

Eine Speicherzelle kann dadurch auf einen bestimmten Wert gesetzt werden, daß ein Register auf diesen Wert gesetzt (siehe dazu Abschnitt 3.1.) und dieser Wert dann abgespeichert wird.

Mit Hilfe der TAS-Sprache können Speicherzellen auf einen Wert gesetzt werden, indem sie durch die Angabe einer Konstanten bei der Übersetzung belegt werden.

Zum Löschen einer Speicherzelle steht der Befehl

LC	n	Lösche Speicher	$\langle n \rangle := +0$ $\langle n \rangle_n := \langle n \rangle_n$
----	---	-----------------	---

zur Verfügung. Die Typenkennung der Speicherzelle bleibt erhalten. Soll die Speicherzelle mit einer bestimmten Typenkennung gelöscht werden, so ist dies z.B. über die Befehle LR und C möglich.

Enthielt die Speicherzelle ein Zahlwort, so bleibt die Markenstelle ebenfalls erhalten.

```

BA 118,
ZTR A3,
C ANTON,

B ((,--DM)),
C ANTON,

A= 'AEG'/R,
EMIL= 128361,

LC ANTON,

LR H3,
CH ANTON,

LC ANTON,
ZT3 ANTON,
    
```

Bild 3.7 Beispiele für Setzen und Löschen von Speicherzellen

3.4. Typenkennung

In einigen Fällen kann es wünschenswert sein, die Typenkennung eines Ganzwortes zu ändern. So liefert z.B. der Befehl BA im Register A stets die Typenkennung 1. Handelt es sich dabei jedoch um Alphatext, so kann es erforderlich sein, diesem Wort die Typenkennung 3 zu geben. Des weiteren ist es z.B. möglich, eine Liste von Alphawörtern mit Typenkennung 3 mit einem Wort abzuschließen, das die Typenkennung 2 hat. Bei den Tabellensuchbefehlen wird die Typenkennung als Abbruchkriterium verwendet. Bei den Befehlen TLI, TLD und TDM müssen alle Wörter der Tabelle die Typenkennung des Suchwortes haben. Bei den Befehlen TMAX und TMIN muß das letzte Wort einer Tabelle eine andere Typenkennung haben als die Wörter der Tabelle. Dieses Wort gehört nicht mehr zur Tabelle.

Die Typenkennung kann sowohl in einem der Rechenwerksregister als auch im Speicher auf einen neuen Wert gesetzt werden. Hierzu dienen die Befehle

ZTR	s	Setze Typenkennung im Register	$\langle s_2 \rangle_t := s_1$
ZT0	n	Setze Typenkennung 0	$\langle n \rangle_t := 0$
ZT1	n	Setze Typenkennung 1	$\langle n \rangle_t := 1$
ZT2	n	Setze Typenkennung 2	$\langle n \rangle_t := 2$
ZT3	n	Setze Typenkennung 3	$\langle n \rangle_t := 3$

s_1 : 0,1,2,3 oder leer
 s_2 : A,Q,H,D oder leer
 s_3 : leer oder M

Es muß jedoch beachtet werden, daß der größte Teil der Befehle in seiner Wirkung von der Typenkennung abhängig ist und ein Alphatext, der z.B. die Typenkennung 1 hat, wie eine Festkommazahl behandelt wird.

Mit dem Befehl ZTR kann auch das Markenregister M auf den Wert L gesetzt werden. Die Spezifikation M kann mit den anderen Spezifikationen gemischt werden oder auch allein stehen.

```

ZTR A3,

ZTR H2M,

ZT2 ANTON,
    
```

Bild 3.8 Beispiele zum Setzen der Typenkennung

3.5. Marke

Zahlwörter können im Speicher markiert sein. Für die Marke ist das Bit Nummer 1 vorhanden. Wird ein Zahlwort in eines der Rechenwerksregister gebracht, so wird beim größten Teil der Befehle das Markenbit dem Vorzeichenbit angeglichen (1. Bit wird gleich dem 2. Bit). In den Rechenwerksregistern A, Q, D und H ist also eine Marke nicht mehr vorhanden. Ob das Wort markiert war, wird im Register M, dem Markenregister festgehalten. Das Markenregister bleibt unverändert, wenn das Wort nicht markiert war. War es markiert, so wird das Register M auf L gesetzt. Nähere Einzelheiten sind dem Abschnitt "Festkommaarithmetik", 2.4., zu entnehmen.

Bei der Beschreibung der Befehle ist jeweils angegeben, ob die Markenstelle berücksichtigt wird.

Es gibt Befehle, die das Markenbit in der Speicherzelle setzen und löschen und die das Markenbit im Register M setzen und löschen.

3.5.1. Setzen und Löschen im Speicher

Im Speicher kann ein Zahlwort nachträglich mit dem Befehl

ZMC	n	Setze Marke im Speicher	$\langle n \rangle_n := L$
-----	---	-------------------------	----------------------------

markiert werden. Der übrige Teil des Wortes bleibt unverändert. Mit dem Befehl

LMC	n	Lösche Marke im Speicher	$\langle n \rangle_n := 0$
-----	---	--------------------------	----------------------------

kann die Markenstelle gelöscht werden.

Die meisten Speicherbefehle speichern die Zahlwörter unmarkiert ab. Bei den Befehlen

CMT	n	Speichere markiert	$\langle n \rangle := \langle A \rangle$
GMR	n	Speichere mit Marke aus Register	$\langle n \rangle := \langle A \rangle$
CMC	n	Speichere mit Marke aus Speicher	$\langle n \rangle := \langle A \rangle$

wird jedoch die Marke berücksichtigt. Beim Befehl CMT wird gleichzeitig das Markenbit gesetzt. Beim Befehl CMR wird das Markenbit gesetzt, wenn das Markenregister M gesetzt war und beim Befehl CMC wird beim Abspeichern die ursprünglich in der Speicherzelle vorhandene Marke übernommen.

Beim Löschen einer Speicherzelle, in der ein Zahlwort stand (mit Hilfe des Befehls LC), bleibt die Markenstelle (und die Typenkennung) erhalten. Darüber hinaus ist es mit dem Befehl

LMP	n	Lösche markiert	$\langle n \rangle := +0$ $\langle n \rangle_m := L$
-----	---	-----------------	---

möglich, die Speicherzelle zu löschen und gleichzeitig das Markenbit zu setzen. Dies ist jedoch nur möglich, wenn vorher ein Zahlwort in der Speicherzelle stand.

Da nur Zahlwörter (Typenkennung 0 und 1) eine Markenstelle enthalten, sind die vorstehenden Befehle nur auf diese anzuwenden. Bei einem Nichtzahlwort (Typenkennung 2 oder 3) wird ein Typenkennungs-Alarm (TK-Alarm) gegeben.

3.5.2. Setzen und Löschen des Registers M

Um das Register M auf L zu setzen oder um es zu löschen (auf 0 zu setzen), sind die Befehle

ZTR	M	Setze Typenkennung im Register	$\langle s_2 \rangle_t := s_1$
LA	M	Lösche in A	$\langle A \rangle_s := 0$ $\langle M \rangle := 0$ nur bei $s = M$

vorhanden. Bei den Befehlen kann als Spezifikation M angegeben werden.

Außerdem wird bei den Sprungbefehlen SM und SMN, mit denen das Markenregister M abgefragt wird, das Markenregister stets auf 0 gelöscht.

3.6. Merklichter

Dem Programmierer stehen mit dem Register K (Merklichter) 8 Merklichter zur Verfügung, in denen er Binärentscheidungen abspeichern kann. Die Merklichter werden von 1 bis 8 numeriert. Mit dem Befehl

LZL	s_L	s_R	Lösche und setze Merklichter	$\langle K \rangle_s := 0$ $\langle K \rangle_s := L$
-----	-------	-------	------------------------------	--

s: Merklichter
0,1,2,3,4,5,6,7 und 8
0 bedeutet kein Merklichter
(0 muß angegeben werden)

können gleichzeitig Merklichter gelöscht und gesetzt werden. Es können mehrere Merklichter angegeben werden. Zuerst werden die im rechten Teil angegebenen Merklichter gelöscht und danach die im linken Teil angegebenen Merklichter gesetzt. Soll kein Merklichter gesetzt bzw. gelöscht werden, so ist 0 einzutragen. Die Nummern für die Merklichter müssen in beiden Teilen ohne Zwischenraum geschrieben werden. Zwischen den beiden Teilen muß ein Zwischenraum sein. Mit dem Befehl

NL	s	Negiere Merklichter	$\langle K \rangle_s := \langle K \rangle_s$ invertiert
----	---	---------------------	---

s: Merklichter
0,1,2,3,4,5,6,7 und 8
0 bedeutet kein Merklichter
(0 muß angegeben werden)

können Merklichter invertiert werden.

Die Merklichter werden zur Programmverzweigung benutzt und können mit den Befehlen SL, SLL, SLN und SNL abgefragt werden. Dabei kann damit, je nachdem, ob die Sprungbedingung erfüllt ist, ein Löschen der Merklichter verbunden sein.

LZL	248	15,
LZL	0	26,
NL	34,	

Bild 3.9 Beispiele zum Befehl LZL und NL

3.7. Unterprogrammregister

Wird der Befehl SU oder SUE zum Sprung in ein Unterprogramm verwendet, so muß vorher mit dem Befehl

ZU	i	Setze Unterprogrammregister	$\langle U \rangle := i$
----	---	-----------------------------	--------------------------

das Unterprogrammregister U mit einer Indexadresse besetzt werden. In der darauffolgenden Indexzelle wird beim Unterprogrammprung die technische Rücksprungadresse abgespeichert.

Wird kein SU- oder SUE-Befehl verwendet, so ist es zweckmäßig, das Register U auf den Wert 254 zu setzen. In diesem Fall würde ein Alarm gegeben, wenn trotzdem einer der beiden Befehle verwendet würde. Im anderen Fall kann auf den Wert 255 gesetzt werden. Die technische Rücksprungadresse wird dann in der Indexzelle 0 und ggf. in den darauf folgenden abgespeichert. Die Indexzellen müssen natürlich mit dem Pseudobefehl INDEX freigehalten werden.

Näheres dazu ist dem Abschnitt "Teilprogramme" zu entnehmen.

Vor dem Start eines Operators muß mit dem Pseudobefehl UNTPR das Register U vorbesetzt werden. Dieses hat die gleiche Wirkung wie eine Besetzung durch den Befehl ZU.

Des weiteren wird das Register U auch neu gesetzt, wenn der Befehl BCI verwendet wird (siehe Abschnitt 3.8.).

UNTPR	254,
UNTPR	255,
INDEX	(3),
ZU	254,
ZU	255,
INDEX	0(3),
ZU	15,
INDEX	16(2),

Bild 3.10 Setzen des Registers U

3.8. Indexbasis

Der Inhalt des Registers X (Indexbasisregister) gibt an, bei welcher Adresse der Indexbereich beginnt. Vor dem Start eines Operators muß mit dem Befehl XBASIS angegeben werden, auf welchen Wert das Register X zu setzen ist. Mit dem Befehl

ZI	m	Setze Indexbasis	$\langle X \rangle := \langle m \rangle_{3-24}$
----	---	------------------	---

kann das Register X auf einen anderen Wert gesetzt werden. Der Adressenteil des Befehls gibt an, in welchem Halbwort der neue Befehl steht. Der neue Wert kann auch als Literal im Adressenteil stehen. Eine ähnliche Wirkung hat der Befehl

BCI	n	Bringe und speichere Indexbasis	$\langle X \rangle := \langle n \rangle_{3-24}$ $\langle U \rangle := \langle n \rangle_{41-48}$
-----	---	---------------------------------	---

Bei ihm wird außerdem noch das Register U neu gesetzt. Des weiteren werden die alten Registerstände von X und U abgespeichert. Die neuen Registerstände stehen in einem Ganzwort. Die Adresse dieses Ganzwortes steht im Adressenteil des Befehls BCI. Die Registerstände und der Inhalt des Ganzwortes werden also durch den Befehl BCI gegeneinander ausgetauscht.

X =	XBASIS	X,
	DSP	256,
	ZI	(X/A),
X =	ASP	100,
X =	DSP	50,
XADR =	X/AGV, 0/HV,	
	BCI	XADR,
X =	DSP	50,
	BCI	(XADR=X/AGV,0/HV),
	BCI	XADR,

Bild 3.11 Setzen des Registers X

4. TABELLEN DURCHSUCHEN

Mit Hilfe der Tabellensuchbefehle ist es möglich, mit einem Befehl eine Tabelle zu durchsuchen, die aus einer beliebigen Anzahl von Ganzwörtern besteht. Der Anfang der Tabelle wird durch die Adresse angegeben, die im Adressenteil des Befehls steht. Das Ende der Tabelle wird (außer TLOG) durch einen Wechsel der Typenkennung gekennzeichnet. Das bedingt, daß in der Tabelle die Wörter, die durchsucht werden sollen, alle die gleiche Typenkennung haben müssen.

Eine Tabelle kann nach folgenden Kriterien durchsucht werden:

- Ist ein Wort identisch mit einem Suchwort, das im Register D vorgegeben ist
- Ist ein Wort gleich oder größer einem im Register D vorgegebenen Suchwort
- Das größte oder das kleinste Wort der Tabelle wird gesucht.

Daneben gibt es noch die Möglichkeit der Dehnung des Suchvorgangs, d.h. es wird nicht jedes Wort der Tabelle durchsucht, sondern jedes zweite, dritte, vierte, usw. Der Dehnungswert steht im Register B und gibt die Adressendifferenz zweier aufeinanderfolgender Wörter an. Da Ganzwörter untersucht werden, ist, wenn z.B. jedes vierte Wort der Tabelle verglichen werden soll, der Wert 8 einzusetzen.

Des weiteren ist es möglich, beim Vergleich des Wortes nicht das ganze Wort zu vergleichen, sondern nur einen Teil des Wortes. Hierzu dient eine Maske, die im Register H bereitzustellen ist. Es werden nur die Binärstellen eines Wortes verglichen, bei denen in der Maske das Bit auf 0 gesetzt ist. Es wird der dem "Null-Feld" der Maske entsprechende Teil der Wörter der Tabelle verglichen.

Eine besondere Stellung nimmt der Befehl TLOG ein. Er durchsucht eine Tabelle logarithmisch. Das bedeutet, daß die Suchzeiten sehr kurz werden. Um dies zu erreichen, muß die Tabelle mit aufsteigenden Werten geordnet sein. Im Bild 4.1 sind die Rechenzeiten der Tabellensuchbefehle in einem Diagramm angegeben.

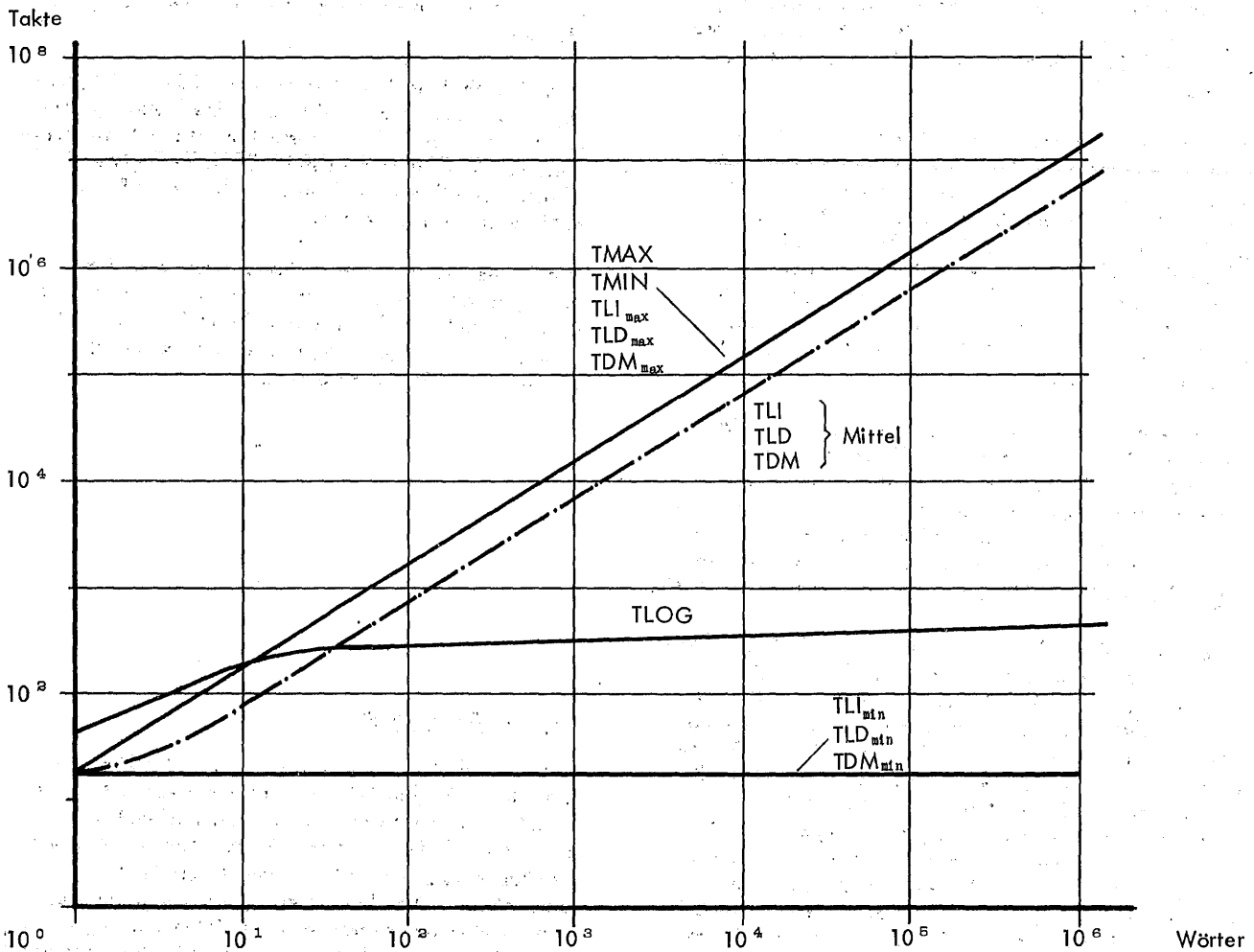


Bild 4.1 Rechenzeiten der Tabellensuchbefehle

4.1. Durchsuchen auf Identität

Um eine Tabelle auf einen Wert zu durchsuchen, der identisch ist mit dem Suchwort, steht der Befehl

TLI	n	Tabelle durchsuchen auf Identität	$\langle n+2k \rangle = \langle D \rangle$
-----	---	-----------------------------------	--

k: 0,1,2,3,...

zur Verfügung.

Das Suchwort muß vor der Ausführung des Befehls TLI ins Register D gebracht werden.

Die Tabelle beginnt bei der im Adressenteil des Befehls angegebenen Adresse und besteht nur aus Wörtern, die die gleiche Typenkennung haben wie das Suchwort im Register D. Wird ein Wort gefunden, das eine andere Typenkennung hat, wird der Suchvorgang abgebrochen, und die Ausführung des Befehls ist beendet. Im Register B steht dann die Adresse des ersten Wortes hinter der Tabelle, d.h. das Wort mit der anderen Typenkennung. Es wird ein Typenkennungsalarm gegeben, der mit dem Befehl SAT abzufangen ist.

Der Suchvorgang beginnt am Tabellenanfang. Das erste Wort, das mit dem Suchwort identisch ist, beendet den Suchvorgang und damit die Ausführung des Befehls TLI. Das gefundene Wort wird ins Register A gebracht (Register A und Register D sind damit identisch). Die Adresse des (zuerst) gefundenen Wortes wird in das Register B gebracht.

Für den Vergleich werden die Wörter der Tabelle ins Register A gebracht (dabei wird bei Zahlwörtern - wie üblich - das erste Bit dem zweiten angeglichen). Der Vergleich wird durchgeführt wie bei dem Befehl SI (Springe wenn identisch), d.h. alle Binärstellen müssen identisch sein. Bei Zahlwörtern ist die positive Null identisch mit der negativen Null. Näheres dazu siehe im Abschnitt 2. Sprungbefehle. Ist bei Gleitkommazahlen das Suchwort nicht normalisiert, so wird es vor dem Suchvorgang normalisiert.

Ist es nicht sicher, ob ein identisches Wort in der Tabelle vorhanden ist, so ist auf jeden Fall der Befehl SAT (Springe wenn Alarm (Typenkennung)) anzuschließen. Im einfachsten Fall kann das der Befehl "SAT IR" sein, d.h. ein Sprung auf den nächsten Befehl. Es kann jedoch auch auf einen anderen Programmteil gesprungen werden, der den Fall, daß kein Wort gefunden wurde, weiterbehandelt.

Im Bild 4.2 ist ein Beispiel für den Befehl TLI angegeben. Die Tabelle ist 500 Ganzwörter lang. Sie müssen die Typenkennung 3 haben. Abgeschlossen wird die Tabelle mit einem Wort mit der Typenkennung 2 (es hat hier den Wert 0). Es wird gesucht, ob ein Wort vorhanden ist, bei dem die linken drei Oktaden die Zeichen END enthalten und die rechten drei Oktaden "Null" sind. Wird ein Wort gefunden, so wird im Programm fortgefahren (die Adresse des gefundenen Wortes steht dann im Register B), im anderen Fall wird auf die Adresse "KWORT" gesprungen (im Register B steht dann die Adresse "TABELLE +1000"). Mit dem Befehl ASP wurde nur Platz freigehalten für 500 Ganzwörter. Die Tabelle ist durch ein anderes Programmstück zu füllen, bevor der Befehl TLI angewendet wird.

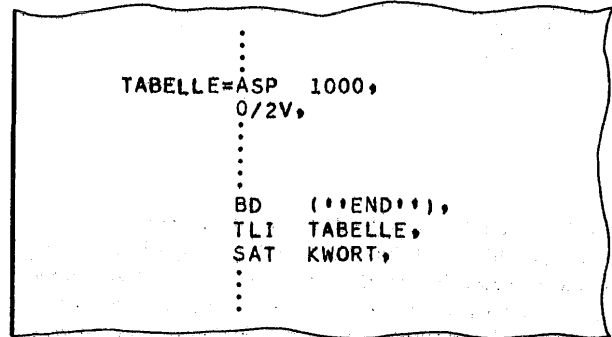


Bild 4.2 Beispiel für den Befehl TLI

Eine ähnliche Wirkung wie der Befehl TLI hat der Befehl

TDM	n	Tabelle durchsuchen mit Dehnung und Maske	$\langle n+k \langle B \rangle \rangle_x = \langle D \rangle_x$ für $\langle H \rangle_x = 0$
-----	---	---	--

k: 0,1,2,3,...

Eine Verarbeitung von Zahlwörtern ist mit diesem Befehl jedoch eingeschränkt. Die Wörter der Tabelle werden stets als Bitmuster betrachtet. Die Markenstelle eines Zahlwortes bleibt also auch beim Vergleich die Markenstelle, und positive und negative Null sind nicht mehr identisch.

Dafür besteht beim Befehl TDM zusätzlich die Möglichkeit, mit Dehnung zu arbeiten und eine Maske zu benutzen. Bei diesem Befehl besteht damit die Notwendigkeit, vor Ausführung des Befehls TDM das Register B (Dehnungswert) und das Register H (Maske) auf einen definierten Wert zu setzen.

Die Adresse des gefundenen Wortes bzw. wenn kein Wort gefunden wurde, das erste Wort mit anderer Typenkennung, steht wie beim Befehl TLI im Register B. Welchen Inhalt die Register A, Q und D haben, ist dem Abschnitt 4.1.2. zu entnehmen.

4.1.1. Mit Dehnung

Beim Befehl TDM ist es möglich, nicht alle Wörter der Tabelle zu durchsuchen, sondern einen Abstand anzugeben, der zwischen einzelnen zu durchsuchenden Wörtern bestehen soll. Es kann also z.B. nur jedes 4. Ganzwort der Tabelle auf Identität untersucht werden. Wir sprechen hier vom Tabellensuchen mit Dehnung. Der Dehnungswert muß vor Ausführung des Befehls im Register B angegeben werden. Der Dehnungswert ist die Differenz der Adressen zweier aufeinanderfolgender Wörter, die untersucht werden sollen.

Wird der Dehnungswert 2 angegeben, so wird, wie beim Befehl TLI jedes Wort untersucht. Soll z.B. jedes 4. Ganzwort untersucht werden, so ist der Dehnungswert 8 anzugeben.

Der Dehnungswert kann auch negativ sein. Es ist dann die Endadresse der Tabelle anzugeben. Zu dieser wird dann jeweils der (negative) Dehnungswert addiert. Dadurch wird die Tabelle, vom Ende beginnend, in Richtung des Anfangs hin durchsucht.

Beim Dehnungswert 0 wirkt der Befehl TDM wie der Befehl NULL. Ist der Dehnungswert ungerade, so ist die Schrittweite abwechselnd um 1 vermindert und um 1 erhöht, da die Anfangsadresse laufend um den Dehnungswert erhöht wird, zur Speicheransteuerung bei ungerader Adresse aber der um 1 verminderte Wert benutzt wird.

Soll beim Befehl TDM das ganze Wort zum Vergleich herangezogen werden, so wird keine Maske benötigt. Es muß dann das Register H auf den Wert Null gesetzt werden. Nach der Ausführung des Befehls steht dann im Register A ebenfalls Null (siehe Abschnitt 4.1.2.).

Bei der Dehnung ist zu beachten, daß in den angegebenen Schritten die Wörter der Tabelle durchsucht werden. Der letzte Schritt, der aus der Tabelle herausführt, muß auf ein Wort mit anderer Typenkennung führen. Der Inhalt dieses Wortes darf also nicht der Zufälligkeit überlassen sein. Im Bild 4.4 wäre z.B. ein Dehnungswert von 14 nicht möglich. Es müßte dann z.B. heißen

TABELLE = ASP 1008,
0/2V,

Damit würde der letzte (72.) Schritt auf jeden Fall auf das Wort mit Typenkennung 2 kommen.

4.1.2. Mit Maske

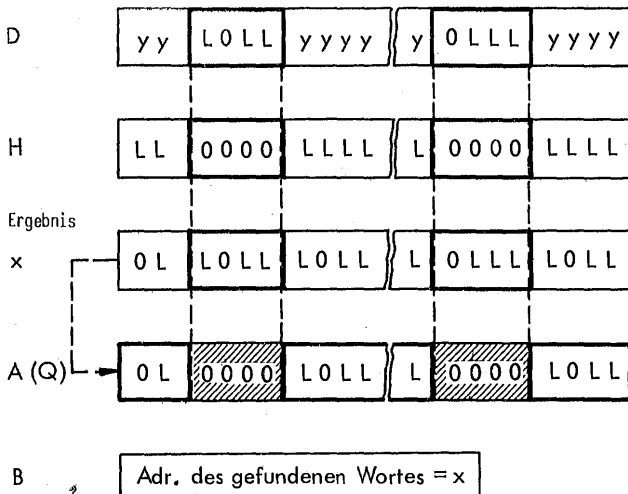
Beim Befehl TDM kann eine Maske angegeben werden. Sie dient dazu, nur bestimmte Teile eines Wortes auf Identität zu vergleichen, und zwar werden alle Binärstellen verglichen, bei denen in der Maske im Register H eine 0 steht (Nullfeld der Maske). Der dem L-Feld der Maske entsprechende Teil wird nicht in den Vergleich einbezogen.

Wird mit Maske gearbeitet, so steht nach Ausführung des Befehls im Register D (Suchwort) der Teil des gefundenen Wortes, der verglichen wurde, also der dem Null-Feld der Maske entsprechende Teil des gefundenen Wortes (oder des ersten Wortes mit anderer Typenkennung). In den Registern A und Q ist der andere Teil des Wortes, das ist also der dem L-Feld der Maske entsprechende Teil. Die anderen Stellen der Register A, Q und D sind jeweils Null.

Im Bild 4.3 wurde an Hand eines Beispiels das Ergebnis eines TDM-Befehls gegeben. Der Einfachheit halber wurden im Beispiel nicht alle Stellen des Ganzwortes niedergeschrieben. Es wurde ein Wort gefunden, das im Null-Feld der Maske identisch ist mit dem Suchwort. Es hat die Adresse x.

Wird beim Arbeiten mit der Maske eine Dehnung nicht gewünscht, so muß das Register B auf 2 gesetzt werden.

Suchwort und Maske



Suchwort
y = beliebig

Maske, 0-Feld stark umrandet

Gefundenes Wort, im Nullfeld mit D identisch

Teile des Wortes x entspr. L-Feld Null-Feld gelöscht

Bild 4.3 Beispiel zum Befehl TDM

N

TAS - Handbuch

Dez. 68

4.1.3. Mit Dehnung und Maske

Beim Befehl TDM kann mit Maske und Dehnung gleichzeitig gearbeitet werden. Es ist entsprechend der Beschreibung in den vorstehenden Abschnitten vor der Ausführung des Befehls TDM die Maske ins Register H zu bringen und der Dehnungswert ins Register B.

Im Bild 4.4. ist ein Beispiel gezeigt, bei dem eine Tabelle von 500 Ganzwörtern vorhanden ist. Davon soll jedes 5. Wort, also insgesamt 100 Wörter, untersucht werden. Durch die Maske wird bestimmt, daß nur die linke Hälfte des Wortes verglichen wird. Es wird ein Wort gesucht, das in der linken Hälfte die Oktaden E, N und D hat.

```

TABELLE=ASP 1000,
0/2V,
:
:
BD (('END')),
BH (('FFFFFF')),
XBA 10,
TDM TABELLE,
SAT KWORT,
:
:

```

Suchwort
Maske
Dehnung

Bild 4.4 Beispiel zum Befehl TDM

Das Suchwort steht im Register D. Falls es eine nicht-normalisierte Gleitkommazahl ist, wird sie normalisiert. Die Wörter der Tabelle, die verglichen werden sollen, müssen die gleiche Typenkennung haben, wie das Suchwort. Hat ein Wort eine andere Typenkennung, wird der Befehl beendet, und es wird, wie beim Befehl TLI, Typenkennungsalarm gegeben, der mit dem Befehl SAT abgefangen werden muß.

Der Vergleich wird so vorgenommen wie beim Befehl SGG. Dabei sind positive und negative Null gleich. Weitere Einzelheiten zum Vergleich sind dem Abschnitt 2. zu entnehmen.

Vom zuerst gefundenen Wort, das der Suchbedingung genügt, wird die Adresse in das Register B gebracht. Im Register A steht nach Ausführung des Befehls das Suchwort. Das gefundene Wort kann z.B. mit dem Befehl MAB B 0 ins Register A gebracht werden.

```

TABELLE=ASP 2000,
0/0V,
:
:
XBA 20,
BD (10000),
TLD TABELLE,
SAT KWORT,
MAB B 0,
:
:

```

Bild 4.5 Beispiel zum Befehl TLD

4.2. Durchsuchen auf größer oder gleich

Für das Durchsuchen einer Tabelle auf einen Wert, der gleich oder größer ist als das Suchwort, stehen die Befehle TLD und TLOG zur Verfügung. Der letztere wird im Abschnitt 4.3. behandelt. Mit dem Befehl

TLD	n	Tabelle durchsuchen mit Dehnung	$\langle n+k\langle B \rangle \rangle \geq \langle D \rangle$
-----	---	---------------------------------	---

k: 0,1,2,3,...

wird eine Tabelle durchsucht, die bei der Adresse n beginnt. Im Register B muß ein Dehnungswert angegeben werden. Er hat den Wert 2, wenn jedes Wort durchsucht werden soll. In diesem Fall wird also ohne Dehnung gearbeitet. Der Dehnungswert gibt die Differenz der Adressen der aufeinanderfolgenden zu vergleichenden Wörter an. Da Ganzwörter durchsucht werden, ist der Dehnungswert ganzzahlig. Weitere Einzelheiten zum Dehnungswert sind bereits im Abschnitt 4.1.1. beschrieben.

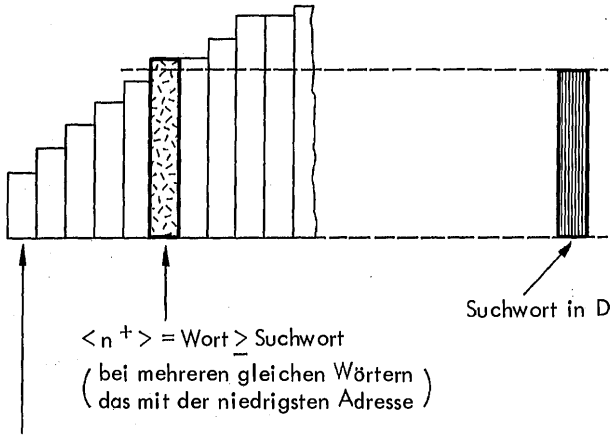
4.3. Logarithmisch durchsuchen

Mit dem Befehl

TLOG	n	Tabelle durchsuchen logarithmisch	$\langle n+2k \rangle_x \geq \langle D \rangle_x$ für $\langle H \rangle_x = 0$
------	---	-----------------------------------	--

k: 0,1,2,3,...

lassen sich Tabellen in kürzester Zeit durchsuchen. Zu diesem Zweck muß die Tabelle nach aufsteigenden Werten geordnet sein. Der erste Wert muß stets kleiner sein als das Suchwort, und jedes folgende Wort muß gleich oder größer sein als das vorhergehende.



1. Wort der Tabelle < Suchwort
 Bild 4.6 Beispiel einer Tabelle zum Befehl TLOG

Die Länge der Tabelle wird bei diesem Befehl nicht durch den Wechsel der Typenkennung begrenzt, sondern muß im Register A vor Ausführung des Befehls in Ganzwörtern angegeben werden. Sie muß mindestens 2 und darf höchstens 2^{20} Ganzwörter lang sein.

Das Suchwort muß vor Ausführung des Befehls im Register D stehen. Es muß größer als Null sein. Im Register H muß eine Maske vorgegeben werden. Sie bestimmt, welcher Teil des Wortes verglichen wird. Der dem L-Feld der Maske entsprechende Teil des Wortes ist ohne Bedeutung; der dem Null-Feld der Maske entsprechende Teil des Wortes wird verglichen.

Beim Vergleich wird das Wort stets als Bitmuster betrachtet. Er ist damit unabhängig von der Typenkennung. Für Zahlwörter ist der Befehl daher nur bedingt anwendbar. Der Teil des Wortes, der durch das Null-Feld der Maske herausgeblendet wird, wird wie eine positive Festkommazahl betrachtet, d.h. mit anderen Worten, das linksstehende Bit hat stets einen größeren Wert als das rechtsstehende. Damit ist ein Vergleich bei Gleitkommazahlen kaum möglich und bei Festkommazahlen nur, wenn sie stets positiv sind, das Komma an der gleichen Stelle haben und nicht markiert sind.

Verglichen wird, ob die Wörter der Tabelle gleich oder größer sind als das Suchwort. Im Gegensatz zum Befehl TLD wird hier nicht das erste Wort, das die Bedingung erfüllt, genommen, sondern es wird von allen Wörtern, die dieser Suchbedingung genügen, stets das mit der niedrigsten Adresse genommen. Die Adresse dieses Wortes wird ins Register B gebracht.

Von dem gefundenen Wort wird der dem L-Feld der Maske entsprechende Teil ins Register A und ins Register Q gebracht, die anderen Binärstellen sind Null. Im Register H sind die dem L-Feld der Maske entsprechenden Stellen auf Null gelöscht.

Wird kein Wort gefunden, d.h. sind alle Wörter (bzw. der dem Null-Feld entsprechende Teil) kleiner als das Suchwort, so wird ein Bereichsüberschreitungsalarm (BÜ-Alarm) gegeben. Er muß mit dem Befehl SAA (Springe wenn arithmetischer Alarm) abgefangen werden. Die Register \bar{A} und Q werden in diesem Fall mit Typenkennung 3 auf Null gesetzt.

Ist das gefundene Wort größer als das Suchwort oder wird kein Wort gefunden, so wird nicht der nächste Befehl im Programm ausgeführt, sondern der übernächste. Damit ist eine dreifache Verzweigung möglich:

- gefundenes Wort = Suchwort: Der nächste Befehl wird ausgeführt.
- gefundenes Wort größer als Suchwort: Der übernächste Befehl wird ausgeführt.
- kein Wort wird gefunden: Der übernächste Befehl wird ausgeführt und es gibt BÜ-Alarm.

Im Bild 4.7 ist ein Beispiel zum Befehl TLOG aufgeführt. Im Bild 4.8 sind Beispiele von Tabellen aufgezeigt. Die mit dem Programm aus Bild 4.7 gefundenen Wörter sind jeweils unterstrichen. Im rechten Fall wurde kein Wort gefunden.

```

TABELLE=ASP 1000,
:
:
:
BA 500,
BH ('FFFFFF'/L),
BD (128),
TLOG TABELLE,
S GLEICH,
SAA KWORT,
GROESSER=:
:
:
:

```

Länge
 Maske
 Suchwort



Bild 4.7 Beispiel zum Befehl TLOG

TABELLE			
12	127	13	1
25	127	14	2
25	<u>130</u>	15	3
30	<u>130</u>	20	4
100	200	101	5
<u>128</u>	281	<u>205</u>	...
128	500	<u>230</u>	...
150	500	501	...
300	501	700	...
...	101
...	102
...	103

Das nach dem Beispiel aus Bild 4.7 jeweils gefundene Wort ist unterstrichen

Bild 4.8 Beispiele für Tabellen zum Befehl TLOG

TAS - Handbuch

Dez. 68

4.3.1. Prinzip des Suchvorganges

Beim Suchvorgang wird zuerst in die Mitte der Tabelle gesprungen. Ist das Wort kleiner, so liegt das zu suchende Wort in der rechten Hälfte, und es wird dort wiederum in die Mitte gesprungen. Im anderen Fall wird in die Mitte der linken Hälfte gesprungen usw.

Aus diesem Prinzip heraus fällt nach dem ersten Vergleich die Hälfte der Tabelle für den Vergleich weg und danach von der jeweils verbleibenden Hälfte wiederum die Hälfte. Dadurch werden sehr kurze Suchzeiten erreicht.

Wird bei diesem Suchvorgang ein Wort gefunden, das kleiner ist als das vorhergehende, so wird der Suchvorgang abgebrochen und das "vorhergehende" Wort als gefundenes Wort betrachtet. Eine Veränderung der Wirkung ergibt sich dadurch nicht. Dies kann dazu ausgenutzt werden, um eine Dehnung zu erreichen. Im Abschnitt 4.3.2. ist dies beschrieben.

4.3.2. Mit Dehnung

Aus dem Prinzip des Suchvorganges (wie im Abschnitt 4.3.1. beschrieben) ergibt sich die Möglichkeit der Dehnung. Dazu müssen folgende Bedingungen beachtet werden.

- Eine Dehnung kann nur um eine Zweier-Potenz erfolgen, d.h. der Dehnungswert (wie aus den anderen Tabellensuchbefehlen bekannt) ist $2 \cdot 2^x$. Die Wörter, die verglichen werden sollen, wollen wir Kopfwörter nennen. Sind m Kopfwörter vorhanden, so muß die Länge der Tabelle $m \cdot 2^x$ Ganzwörter sein.

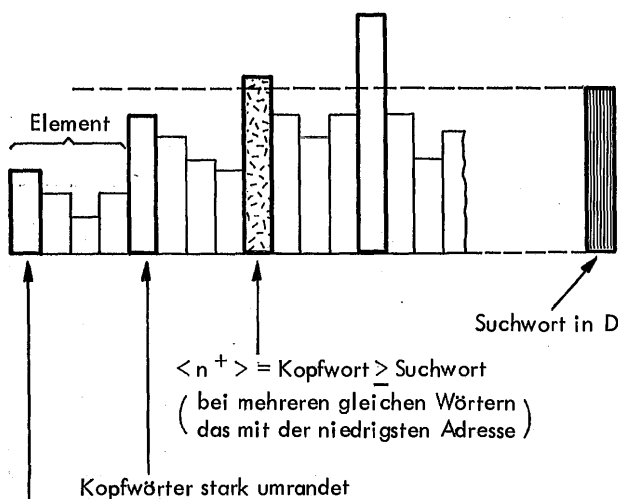
Die zwischen den Kopfwörtern liegenden Wörter müssen stets gleich oder kleiner sein, als das kleinere der sie begrenzenden Kopfwörter; ihre Anzahl ergibt sich zu $2^x - 1$, wenn die Dehnung 2^x Ganzwörter ist.

Vom Befehl TLOG werden dann nur die Kopfwörter untersucht. Sie müssen den bereits im Abschnitt 4.3. beschriebenen Bedingungen entsprechen.

- Das erste Kopfwort muß kleiner sein als das Suchwort.
- Jedes folgende Kopfwort muß gleich oder größer sein als das vorhergehende.

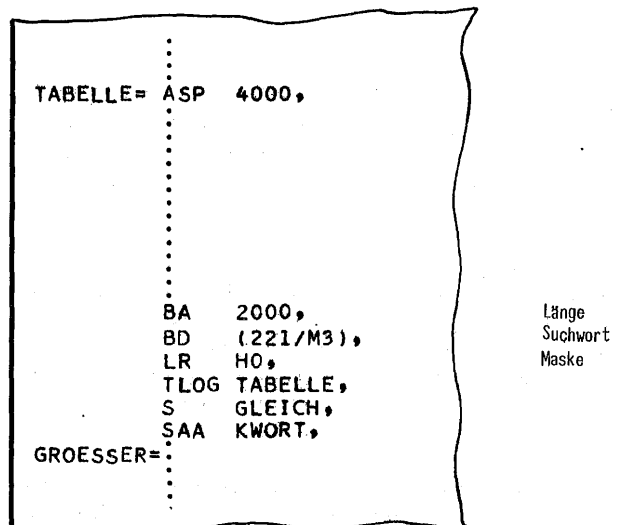
Werden die in diesem Abschnitt genannten Bedingungen eingehalten, so wird, ohne daß die Dehnung (explizit) angegeben ist, eine Dehnung erreicht.

Beziehen wir die Maske in unsere Betrachtungen mit ein, so beziehen sich alle Betrachtungen in bezug auf gleich, kleiner oder größer als das Suchwort auf den Teil, der durch das Null-Feld der Maske ausgeblendet wird. Soll keine Maske verwendet werden, so ist das Register H zu löschen.



1. Wort = Kopfwort der Tabelle < Suchwort

Bild 4.9 Beispiel einer Tabelle zum Befehl TLOG



Inhalt der Tabelle

25/M,	$2^{4?}$	25	Kopfwort
300,		300	
22,		22	
10 300,		10 300	
50/M,	$2^{4?}$	50	Kopfwort
3 161,		3 161	
15,		15	
25,		25	
200/M,	$2^{4?}$	200	Kopfwort
20,		20	
100 521,		100 521	
225,		225	
<u>225/M,</u>	<u>$2^{4?}$</u>	<u>225</u>	Kopfwort
2 200,		2 200	
⋮		⋮	

Unterstrichen = gefundenes Wort

Bild 4.10 Beispiel für Tabelle zum Befehl TLOG mit Dehnung

Im Bild 4.9 ist das Beispiel einer Tabelle mit Dehnung 4 (2^2) gezeigt. Jedes vierte Wort ist ein Kopfwort. In Bild 4.10 ist ein Beispiel, in dem die Tabelle aus positiven Festkommazahlen besteht. Hier sind mit Hilfe der Marke die Kopfwörter gekennzeichnet. Da der Befehl TLOG nur Bitmuster vergleicht, wirkt die Marke so, als ob die Zahl mit 2^{47} multipliziert wäre. Damit sind die Kopfwörter alle größer als die anderen (nicht markierten) Zahlwörter. Das Suchwort muß im Register D ebenfalls das Markenbit gesetzt haben; das wird dadurch erreicht, daß das Suchwort die Typenkennung 3 erhält und dadurch mit dem Befehl BD unverändert, also mit Markenbit, ins Register D gebracht wird. Der Suchvorgang ist unabhängig von der Typenkennung.

4.4. Durchsuchen auf Maximum oder Minimum

Für das Durchsuchen einer Tabelle auf ihren maximalen bzw. ihren minimalen Wert stehen die Befehle

TMAX	n	Tabelle durchsuchen auf <u>Maximum</u>	$\langle n+k\langle B \rangle \rangle_x = \max.$ für $\langle H \rangle_x = 0$
TMIN	n	Tabelle durchsuchen auf <u>Minimum</u>	$\langle n+k\langle B \rangle \rangle_x = \min.$ für $\langle H \rangle_x = 0$

k: 0,1,2,...

zur Verfügung. Beide Befehle arbeiten mit Maske und Dehnung.

Es wird stets ein Bitmuster verglichen; der Vergleich ist unabhängig von der Typenkennung. Die Verwendung bei Zahlwörtern ist also eingeschränkt.

Alle zu durchsuchenden Wörter der Tabelle müssen die gleiche Typenkennung haben wie das erste Wort der Tabelle. Ein Wort mit anderer Typenkennung bricht den Suchvorgang ab. Ein Suchwort wird nicht benötigt.

Die Maske muß im Register H stehen. Es werden nur die Stellen der Wörter verglichen, die dem Null-Feld der Maske entsprechen.

Im Register B muß der Dehnungswert angegeben sein. Wird kein Dehnungswert gewünscht, so ist der Wert 2 einzusetzen. Weitere Einzelheiten zur Dehnung siehe Abschnitt 4.1.1.

Als Ergebnis der Befehle steht die Adresse des gefundenen Wortes im Register B. Im Register A und im Register Q steht der dem L-Feld der Maske entsprechende Teil des gefundenen Wortes und im Register D der dem Null-Feld der Maske entsprechende Teil des gefundenen Wortes.

Es wird stets ein Wort gefunden. Ein Alarm kann nicht auftreten.

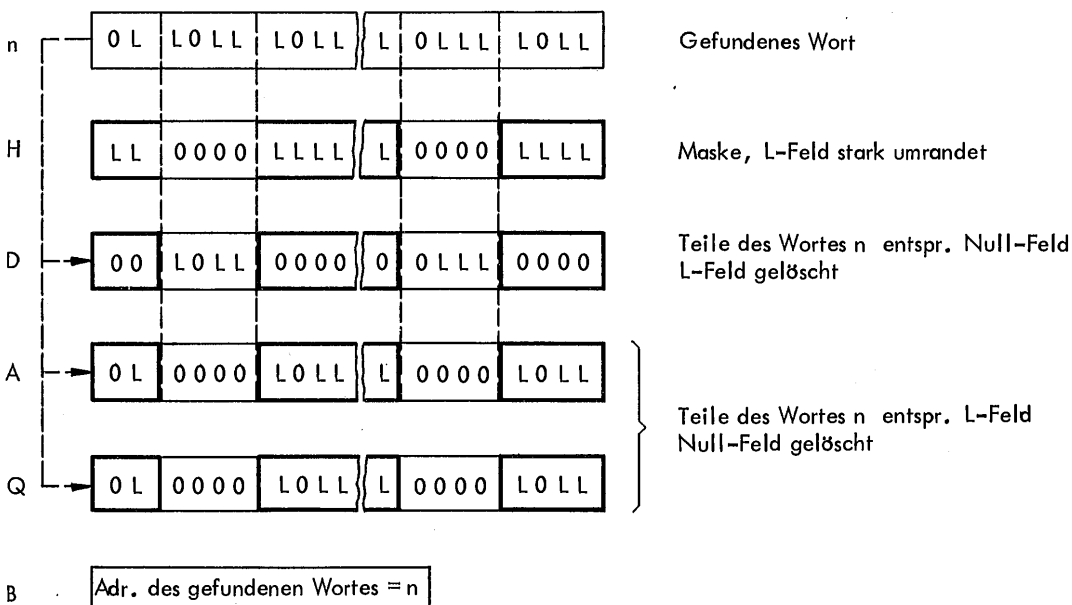


Bild 4.11 Ergebnis beim Befehl TMAX bzw. TMIN

5. WORTGRUPPEN

Größere Gruppen von Ganzwörtern können mit einem Befehl innerhalb des Kernspeichers transportiert werden. Es sind dies die Wortgruppentransporte. Beim Transport kann sowohl am Anfang der Wortgruppe als auch am Ende begonnen werden.

Des weiteren gibt es zwei Befehle, die es ermöglichen, alle Inhalte der Rechenwerksregister (A, Q, D, H, M, Y) und die Register B, K und U des Befehlswerks im Speicher sicherzustellen bzw. die Speicherinhalte wieder in die Register zu bringen.

zur Verfügung. Die beiden Befehle unterscheiden sich darin, daß beim Befehl WTV am Anfang der Speicherbereiche angefangen wird und beim Befehl WTR am Ende der Speicherbereiche.

Vor der Ausführung der Befehle muß in der Indexzelle i_R die Anfangsadresse (bei WTR die Endadresse) des Quellenbereichs und in der Indexzelle i_L die des Zielbereichs stehen. Im Register B muß angegeben sein, wieviel Ganzwörter transportiert werden sollen.

Die Befehle bewirken nun, daß soviel Ganzwörter, wie das Register B angibt, vom Quellenbereich in den Zielbereich unverändert transportiert werden.

Nach der Ausführung der Befehle steht im Register B die Endadresse (bei WTR die Anfangsadresse) des Zielbereichs.

5.1. Wortgruppentransport

Um eine Gruppe von Wörtern von einem Speicherbereich in einen anderen zu bringen, stehen die Befehle

WTV	i_L	i_R	Wortgruppentransport vorwärts	$\langle \langle i_L \rangle + 2k \rangle := \langle \langle i_R \rangle + 2k \rangle$ für $k=0, 1, 2, \dots, \langle B \rangle - 1$ $\langle B \rangle := \langle i_L \rangle + 2(\langle B \rangle - 1)$
WTR	i_L	i_R	Wortgruppentransport rückwärts	$\langle \langle i_L \rangle - 2k \rangle := \langle \langle i_R \rangle - 2k \rangle$ für $k=0, 1, 2, \dots, \langle B \rangle - 1$ $\langle B \rangle := \langle i_L \rangle - 2(\langle B \rangle - 1)$

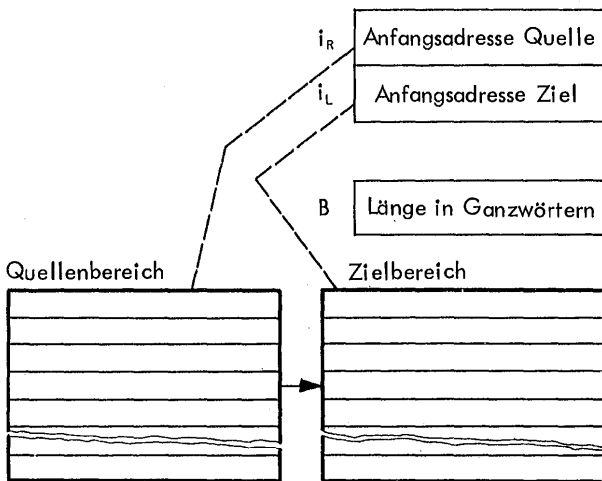


Bild 5.1 Wirkung des Befehls WTV

```

QUELLE = ASP 1000,
ZIEL =   ASP 1000,
        XBA QUELLE,
        XC  QU,
        XBA ZIEL,
        XC  Z,
        XBA 500,
        WTV Z  QU,
    
```

```

QUELLE = DSP 2000,
ZIEL =   DSP 10000,
        TCB (QUELLE + 1998/A),
        XC  QU,
        TCB (ZIEL + 9998/A),
        XC  Z,
        XBA 1000,
        WTR Z  QU,
    
```

Bild 5.2 Beispiele für Wortgruppentransport

N

5.2. Register sicherstellen

In vielen Fällen, z.B. beim Auftreten von Fehlern, kann es nötig sein, die Inhalte der wichtigsten Register sicherzustellen. Hierzu stehen die Befehle

QCR	n	Bequemes Speichern aller Register	Die Register werden unverändert abgespeichert
QBR	n	Bequemes Bringen aller Register	Die Register werden unverändert gebracht

zur Verfügung. Zur Sicherstellung werden 6 Ganzwörter im Speicher benötigt. Im Bild 5.3 ist gezeigt, wie die Register abgespeichert werden.

Mit dem Befehl QCR werden alle Register, bis auf die Register X und F, abgespeichert. Im Adressenteil des Befehls steht die Anfangsadresse des Speicherbereichs. Alle Register werden unverändert abgespeichert.

Mit dem Befehl QBR werden alle Registerinhalte aus dem Speicher wieder zurückgebracht. Beim Befehl QBR muß jedoch im Adressenteil des Befehls die Endadresse des Abspeicherbereichs stehen.

a	2		24	<K>	8	<Y>	8	<U>	8
t _A		<A>							48
t _Q		<Q>							
t _D		<D>							
t _H		<H>							
2		<BT>	24	0					24

a = 2 : <M> = 0
 = 3 : <M> = L
 BT : Prüfregister

Bild 5.3 Speicherbereich für die Befehle QCR und QBR

Mit dem Befehl QCR können also die Registerinhalte sichergestellt werden, und mit dem Befehl QBR kann der alte Zustand wieder hergestellt werden.

Der Wert des Registers X steht gleichzeitig im Leitblock des Prozesses. Er kann, falls erforderlich, von dort mit Hilfe des Befehls

BLEI 4

in die linke Hälfte des Registers A mit Typenkennung 3 geholt werden. Die rechte Hälfte ist durch eine andere Adresse belegt. Die Indexbasisadresse (und evtl. auch die andere Adresse) kann dann z.B. in dem 7. Wort des Speicherbereichs abgespeichert werden. Später kann durch den Befehl ZI (oder BCI) das Register X auf den alten Wert zurückgesetzt werden.

Das Sicherstellen des Registers F hängt von verschiedenen Faktoren ab. Ob und wie es sichergestellt wird, muß von Fall zu Fall entschieden werden.

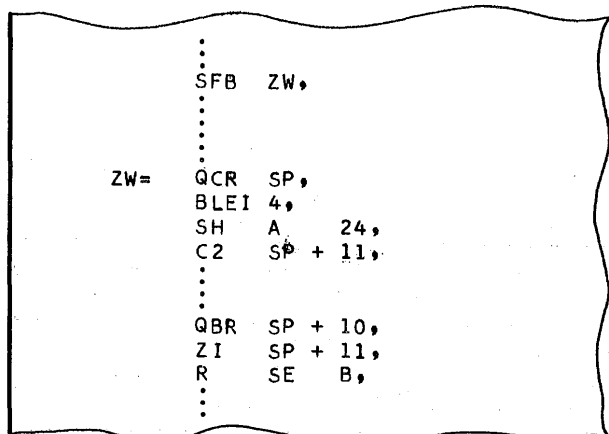
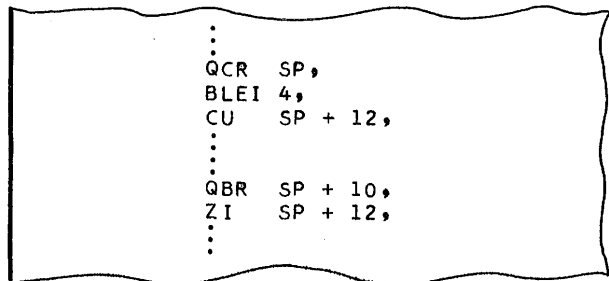


Bild 5.4 Beispiele für Sicherstellen

6. BOOLESCHE OPERATIONEN

Für die nachstehenden Booleschen Verknüpfungen sind Befehle vorhanden.

Benennung			Zeichen
Konjunktion	UND	ET	\wedge
Disjunktion	ODER	VEL	\vee
Antivalenz		AUT	\oplus
Negation			\neg

Die Negation ist bei den Registern A, Q, D und H und bei den Merklichtern möglich. Die anderen Funktionen verknüpfen entweder das Register A mit einer Speicherzelle oder den Adressenteil des Befehls mit dem Register H.

Ist bei der Verknüpfung von Register A und Speicherzelle n die Typenkennung verschieden, so erhält das Ergebnis die größere der beiden Typenkennungen. Bei der Verknüpfung des Adressenteils mit dem Register H erhält das Ergebnis die Typenkennung des Registers H.

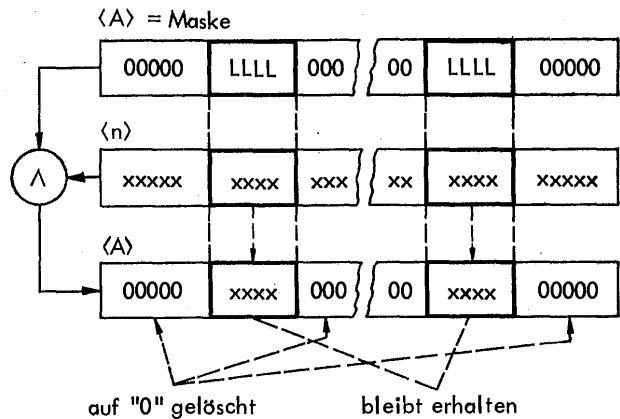
Des weiteren gibt es den Befehl ZUS, der mit Hilfe einer Maske zwei Wörter zusammensetzt.

Als Nebenwirkung steht der Inhalt der Speicherzelle n bzw. der Adressenteil des Befehls im Register D.

zur Verfügung. Während beim Befehl ET der Inhalt des Registers A und der Inhalt der Speicherzelle n miteinander verknüpft werden, wird beim Befehl ETA, der während der Abbruchphase auf 24 Bits erweiterte (und ggf. modifizierte) Adressenteil des Befehls mit dem Inhalt des Registers H verknüpft. In beiden Fällen steht das Ergebnis im Register A.

Die Befehle können auch dazu benutzt werden, um bestimmte Teile aus einem Wort herauszublenken und den restlichen Teil auf 0 zu setzen. Betrachtet man einen der zu verknüpfenden Werte als Maske, so wird von dem anderen Wert der Teil auf Null gelöscht, der dem Null-Feld der Maske entspricht. Der dem L-Feld der Maske entsprechende Teil bleibt erhalten.

Beim Befehl ET kann der Inhalt des Registers A als Maske betrachtet werden. Es wird dann aus dem Inhalt der Speicherzelle der dem L-Feld der Maske entsprechende Teil ins Register A gebracht. Der dem Null-Feld der Maske entsprechende Teil wird auf Null gelöscht.



6.1. Konjunktion

Durch die Konjunktion werden jeweils zwei Binärstellen miteinander verknüpft. Nur wenn beide Binärstellen den Wert "L" haben (die eine UND die andere Binärstelle), tritt als Ergebnis ebenfalls "L" auf. In den anderen Fällen ist das Ergebnis "0". Es ergibt sich also folgende Verknüpfung:

a :=	b	\wedge	c
0	0	0	0
0	0	L	L
0	L	0	L
L	L	L	L

Um zwei Größen konjunktiv miteinander zu verknüpfen, stehen die Befehle

ET	n	ET	$\langle A \rangle := \langle A \rangle \wedge \langle n \rangle$
ETA	z	ET Adressenteil	$\langle A \rangle := \langle H \rangle \wedge z$

z: 0...65 535 (vor Modifizierung)

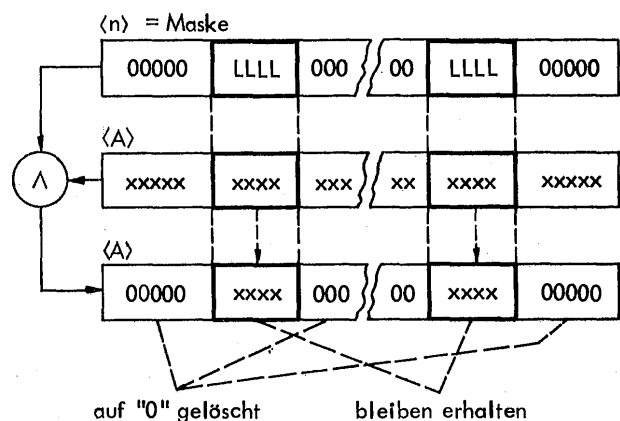


Bild 6.1 Beispiele zum Befehl ET

N

TAS - Handbuch

Dez. 68

Betrachtet man den Inhalt der Speicherzelle n als Maske, so wird im Register A der dem Null-Feld der Maske entsprechende Teil auf 0 gelöscht, während der dem L-Feld der Maske entsprechende Teil erhalten bleibt.

Im Bild 6.1 sind vorstehende zwei Betrachtungsweisen skizziert. Im Bild 6.2 wird im ersten Beispiel aus dem im Register A stehenden Wert von ANTON die vorletzte Tetrade herausgeblendet. Im zweiten Beispiel wird mit der im Register A stehenden Maske aus der Speicherzelle ANTON die vorletzte Tetrade ins Register A gebracht.

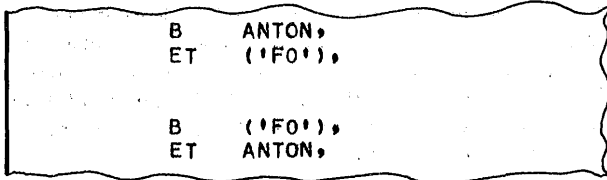


Bild 6.2 Beispiele zum Befehl ET

Beim Befehl ETA kann der im Adressenteil stehende Wert als Maske betrachtet werden. Er bringt aus dem Register H den dem L-Feld des Adressenteils entsprechenden Wert ins Register A. Im Bild 6.3 ist gezeigt, wie mit Hilfe des Befehls ETA die vorletzte Oktade aus dem Register H ausgeblendet und ins Register A gebracht wird. Der Inhalt des Registers H kann zuvor mit dem Befehl BH aus dem Speicher geholt sein.

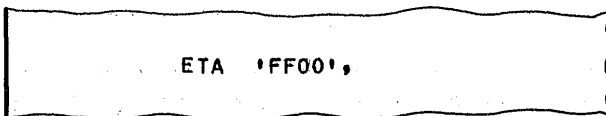
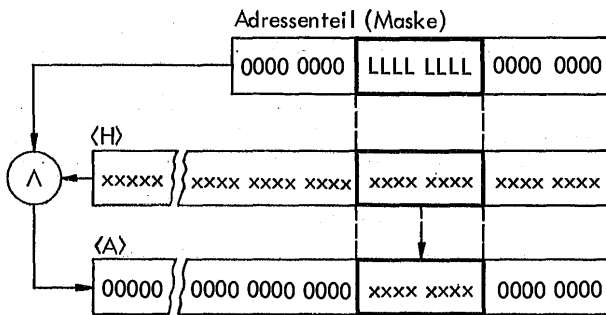


Bild 6.3 Beispiel zum Befehl ETA

Diese Art, mit Hilfe der Maske nur bestimmte Teile eines Wortes zu verarbeiten, wurde bei den Teilwortbefehlen (siehe Abschnitt "Nichtzahlwörter", 10.), den Tabellenbefehlen (siehe Abschnitt 4.) und bei dem Befehl ZUS (siehe Abschnitt 2.5.) angewendet.

6.2. Disjunktion (Adjunktion)

Durch die Disjunktion werden jeweils zwei Binärstellen miteinander verknüpft. Wenn die eine ODER die andere Binärstelle den Wert "L" hat, wird auch das Ergebnis "L". Haben also beide den Wert "0", so wird das Ergebnis "0". Es ergibt sich also folgende Verknüpfung:

a := b	v	c
0	0	0
L	0	L
L	L	0
L	L	L

Um zwei Größen disjunktiv miteinander zu verknüpfen stehen die Befehle

VEL	n	VEL	$\langle A \rangle := \langle A \rangle \vee \langle n \rangle$
VLA	z	VEL Adressenteil	$\langle A \rangle := \langle H \rangle \vee z$

z: 0...65 535 (vor Modifizierung)

zur Verfügung.

Beim Befehl VEL werden der Inhalt des Registers A und der Inhalt der Speicherzelle n miteinander verknüpft und beim Befehl VLA, der während der Abruflphase auf 24 Bits erweiterte (und ggf. modifizierte) Adressenteil des Befehls und der Inhalt des Registers H. Das Ergebnis steht in beiden Fällen im Register A.

Betrachtet man einen der Werte als Maske (wie im Abschnitt 6.1.), so gilt hier, daß der dem 0-Feld der Maske entsprechende Teil erhalten bleibt und der dem L-Feld der Maske entsprechende Teil auf "L" gesetzt wird.

6.3. Antivalenz

Durch die Antivalenz werden jeweils zwei Binärstellen miteinander verknüpft. Haben zwei Binärstellen unterschiedliche Werte, so wird das Ergebnis "L", haben sie gleiche Werte, so wird es "0". Diese Funktion wird auch "exclusives ODER" bezeichnet. Es ergibt sich folgende Verknüpfung:

a := b	≠	c
0	0	0
L	0	L
L	L	0
0	L	L

Um zwei Größen durch die Antivalenz miteinander zu verknüpfen, stehen die Befehle

AUT	n	AUT	$\langle A \rangle := \langle A \rangle \oplus \langle n \rangle$
ATA	z	AUT Adressenteil	$\langle A \rangle := \langle H \rangle \oplus z$

z: 0...65 535 (vor Modifizierung)

zur Verfügung. Beim Befehl AUT werden der Inhalt des Registers A und der Inhalt der Speicherzelle n miteinander verknüpft und beim Befehl ATA der während der Abrufphase auf 24 Bits erweiterte (und ggf. modifizierte) Adressenteil des Befehls und der Inhalt des Registers H. Das Ergebnis steht in beiden Fällen im Register A.

Betrachtet man (wie im Abschnitt 6.1.) einen der Werte als Maske, so gilt hier, daß der dem Null-Feld der Maske entsprechende Teil erhalten bleibt und der dem L-Feld der Maske entsprechende Teil invertiert wird.

6.5. Zusammensetzen

Der Befehl ZUS erlaubt es, zwei Wörter mit Hilfe einer Maske zusammensetzen.

ZUS	n	Setze <u>zusammen</u>	$\langle A \rangle_x := \langle A \rangle_x$: für $\langle H \rangle_x = 0$ $\langle A \rangle_x := \langle n \rangle_x$: für $\langle H \rangle_x = L$
-----	---	-----------------------	--

Vor Ausführung des Befehls muß im Register H die Maske stehen. Der Befehl bewirkt, daß im Register A der Teil erhalten bleibt, der dem Null-Feld der Maske entspricht. Der dem L-Feld der Maske entsprechende Teil wird der Speicherzelle n entnommen und in das Register A eingesetzt. Es werden also der dem Null-Feld der Maske entsprechende Teil des Registers A und der dem L-Feld der Maske entsprechende Teil der Speicherzelle n zu einem neuen Wert im Register A zusammengesetzt.

Sind die Typenkennungen von Register A und der Speicherzelle n unterschiedlich, so erhält das Ergebnis die höhere der beiden Typenkennungen.

Als Nebenwirkung steht der Inhalt der Speicherzelle n im Register D.

6.4. Negation

Die Negation bezieht sich jeweils auf eine Binärstelle. Sie wird in ihrem Wert umgekehrt. Hatte sie den Wert "0", so wird sie auf den Wert "L" gesetzt, hatte sie den Wert "L", so wird sie auf den Wert "0" gesetzt. Es gilt also die Funktion

$a := \neg b$	
0	L
L	0

Für die Negation steht der Befehl

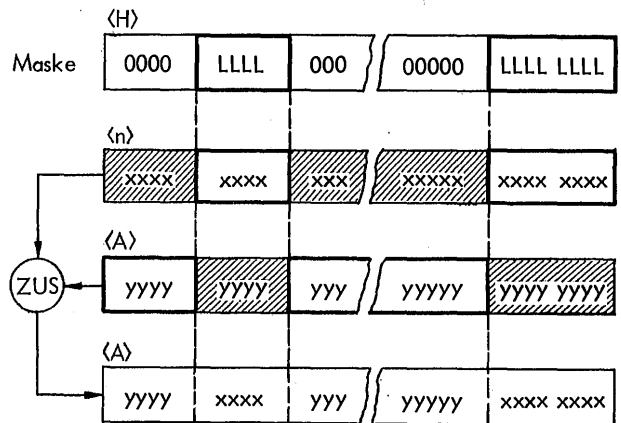
IR	s	Invertiere Register	$\langle s \rangle := \neg \langle s \rangle$
----	---	---------------------	---

s: A, Q, D und H

zur Verfügung. Es können die Rechenwerksregister A, Q, D und H invertiert werden, wobei mehrere angegeben werden können.

Bei Festkommazahlen (Typenkennung 1) bedeutet die Negation, daß die Zahl das Vorzeichen wechselt.

Bei der Typenkennung 0 (Gleitkommazahlen) werden nur die linken 40 Bits invertiert. Damit hat die Gleitkommazahl ihr Vorzeichen gewechselt.



```

:
: BH ('0F00000000FF'),
: ZUS ANTON,
:
:

```

Bild 6.4 Beispiel zum Befehl ZUS



TAS - HANDBUCH

Tabellen, Listen, Formulare

INHALT

<u>ZENTRALCODE ZC1</u>	2N 0812.511
<u>ZEICHENSÄTZE ZS</u>	2N 0810.111
<u>LOCHKARTEN</u>	
Umcodierleistung BINÄR Eingabe	2N 0815.110
Umcodierleistung BINÄR Ausgabe	2N 0815.210
KC1	2N 0813.111
Umcodierleistung KC1 → ZC1	2N 0815.111
Umcodierleistung ZC1 → KC1	2N 0815.211
KC2	2N 0813.112
Umcodierleistung KC2 → ZC1	2N 0815.112
Umcodierleistung ZC1 → KC2	2N 0815.212
KC3	2N 0813.113
Umcodierleistung KC3 → ZC1	2N 0815.113
Umcodierleistung ZC1 → KC3	2N 0815.213
<u>LOCHSTREIFEN</u>	
SC1	2N 0813.211
Umcodierleistung SC1 → ZC1	2N 0815.411
Umcodierleistung ZC1 → SC1	2N 0815.511
SC2	2N 0813.212
Umcodierleistung SC2 → ZC1	2N 0815.412
Umcodierleistung ZC1 → SC2	2N 0815.512
<u>SCHNELLDRUCKER</u>	
DC1	2N 0813.311
Umcodierleistung ZC1 → DC1	2N 0815.311
DC2	2N 0813.312
DC4	2N 0813.314
<u>TASTATURCODE TC1</u>	2N 0813.411
<u>VERSCHLÜSSELTER BINÄRCODE</u>	
VBC1	2N 0811.111
VBC1 auf Lochkarten	2N 0813.151
VBC1 auf Lochstreifen	2N 0813.251
VBC1 auf 9-Spur-Magnetband	2N 0813.651
<u>7-BIT-CODE</u>	2N 0812.411
	Blatt 1 und 2



Informationsverarbeitung
Zentralcode
ZC 1

2N 0812

.511

Einsprüche bis 1. September 1971

Dieser Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG 3: C o d e s, Schriftführer: Böckmann N31/V23, Tel. 2474

1. Zweck und Anwendung

Der hier für den TR 440 festgelegte Zentralcode ZC 1 (8-Bit-Code) ist einer von vielen möglichen Codes.

Er ist der Standardcode des TR 440, auf den sich die Umschlußler der Ein- und Ausgabevermittler und die textverarbeitenden Standardprogramme beziehen.

Der ZC1 enthält alle Zeichen des international genormten 7-Bit-Codes nach DIN 66 003.

2. Code-Tabelle

Der Zentralcode ZC1 ist durch die Code-Tabelle festgelegt (siehe Seite 2).

2.1. Zeichenvorrat und Codierung

Der Zeichenvorrat besteht aus Schriftzeichen und Steuerzeichen. Schriftzeichen sind Ziffern, Buchstaben und Sonderzeichen. Steuerzeichen werden durch eine Gruppe von 2 oder 3 Schriftzeichen dargestellt.

2.2. Bezeichnung der Codierung

Die einzelnen Zeichen des Zentralcodes können durch ihren Dualwert (Binärmuster) angegeben werden.

Jeweils 4 Bits werden zu einer Tetrade zusammengefaßt; die 16 möglichen Werte werden durch Sedezimalziffern dargestellt. Als Sedezimalziffern werden die Ziffern 0 bis 9 und die Buchstaben A bis F verwendet. Um Verwechslungen mit Dezimalwerten zu vermeiden, werden die Sedezimalwerte in Apostrophe eingeschlossen.

Jedes Zeichen kann auch durch den dezimalen Wert der Oktade angegeben werden. Es sind die Werte 0 bis 255 möglich.

Beispiel:

E	&	Zeichen
196	119	Dezimalwert
'C4'	'77'	Sedezimalwert
LL00 0L00	OLLL OLLL	Dualwert

3. Steuerzeichen

3.1. Erläuterungen der Abkürzungen

Soweit die Abkürzungen verwendet werden, die gemäß DIN 66 003 (und ISO) festgelegt sind, ist in Klammern die englische Benennung angegeben.

ACK	Positive Rückmeldung (Acknowledge)
BEL	Klingel (Bell)
BS	Rückwärtsschritt (Backspace)
CAN	Ungültig (Cancel)
CR	Wagenrücklauf (Carriage Return)
DC1	Gerätsteuerung 1 (Device Control)
DC2	Gerätsteuerung 2
DC3	Gerätsteuerung 3
DC4	Gerätsteuerung 4
DEL	Löschen (Delete)
DLE	Datenübertragungsumschaltung (Data Link Escape)
EM	Ende der Aufzeichnung (End of Medium)
ENQ	Stationsaufforderung (Enquiry)
EOT	Ende der Übertragung (End of Transmission)
ESC	Umschaltung (Escape)
ETB	Ende des Datenübertragungsblocks (End of Transmission Block)
ETX	Ende des Textes (End of Text)
FL	Fluchtsymbol
HT	Horizontal-Tabulator (Horizontal Tabulation)
IS1	Informationstrennung 1 (Information Separator)
IS2	Informationstrennung 2
IS3	Informationstrennung 3
IS4	Informationstrennung 4
MZ	Minus Null (Minus Zero)
NAK	Negative Rückmeldung (Negative Acknowledge)
NF	Neues Formular (New Form)
NL	Neue Zeile (New Line)
NL2	Vorschub um 2 Zeilen
NL3	Vorschub um 3 Zeilen
NL4	Vorschub um 4 Zeilen
NL5	Vorschub um 5 Zeilen
NL6	Vorschub um 6 Zeilen
NL7	Vorschub um 7 Zeilen
NUL	Nil (Null)
NU1	Nil 1
NU2	Nil 2
NU3	Nil 3
NU4	Nil 4
NU5	Nil 5
PZ	Plus Null (Plus Zero)
SI	Rückschaltung (Shift-in)
SO	Dauerumschaltung (Shift-out)
SOH	Anfang des Kopfes (Start of Heading)
SP	Zwischenraum (Space)
STX	Anfang des Textes (Start of Text)
SUB	Substitution (Substitute Character)
SYN	Synchronisierung (Synchronous Idle)
TE	Textende
VT	Vertikal-Tabulator (Vertical Tabulation)
VT3	Zeilenvorschub gemäß Spur 3
VT4	Zeilenvorschub gemäß Spur 4
VT5	Zeilenvorschub gemäß Spur 5
VT6	Zeilenvorschub gemäß Spur 6
VT7	Zeilenvorschub gemäß Spur 7
VT8	Zeilenvorschub gemäß Spur 8

3 Ausgaben:
November 69

Diese Unterlage darf weder kopiert, noch
dritten Personen mitgeteilt, noch ander-
weitig mißbräuchlich benutzt werden

Änderungen:
Alle Zeichen des 7-Bit-Codes nach DIN 66 003 im
ZC1 zu haben, wurden die noch fehlenden Steuerzei-
chen eingearbeitet.

Bm

N3/N

N 31/V23

Böckmann

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2 - 6

V

Code-Tabelle

2⁷ 2⁰
1. Sede-
zimale 2. Sede-
zimale

0000	000L	00LO	00LL	0L00	0L0L	0LLO	0LLL	LO00	LO0L	LOLO	LOLL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
⁰ NUL	¹⁶ NL6	³² SUB	⁴⁸ BEL	⁶⁴	⁸⁰	⁹⁶ "	¹¹² %	¹²⁸ ^	¹⁴⁴ +	¹⁶⁰ (¹⁷⁶ 0	¹⁹² A	²⁰⁸ Q	²²⁴ a	²⁴⁰ q	0	0000
¹ NU1	¹⁷ NL5	³³ EM	⁴⁹ DC1	⁶⁵	⁸¹	⁹⁷ ,	¹¹³ §	¹²⁹ v	¹⁴⁵ -	¹⁶¹)	¹⁷⁷ 1	¹⁹³ B	²⁰⁹ R	²²⁵ b	²⁴¹ r	1	000L
² NU2	¹⁸ NL4	³⁴ CAN	⁵⁰ DC2	⁶⁶	⁸²	⁹⁸ /	¹¹⁴ #	¹³⁰ ¬	¹⁴⁶	¹⁶² [¹⁷⁸ 2	¹⁹⁴ C	²¹⁰ S	²²⁶ c	²⁴² s	2	00LO
³ NU3	¹⁹ NL3	³⁵	⁵¹ DC3	⁶⁷	⁸³	⁹⁹ \	¹¹⁵ \$	¹³¹ /	¹⁴⁷	¹⁶³]	¹⁷⁹ 3	¹⁹⁵ D	²¹¹ T	²²⁷ d	²⁴³ t	3	00LL
⁴ NU4	²⁰ NL2	³⁶	⁵² DC4	⁶⁸	⁸⁴	¹⁰⁰	¹¹⁶ ¢	¹³²	¹⁴⁸	¹⁶⁴	¹⁸⁰ 4	¹⁹⁶ E	²¹² U	²²⁸ e	²⁴⁴ u	4	0L00
⁵ NU5	²¹ NL	³⁷ TE	⁵³ FL	⁶⁹	⁸⁵	¹⁰¹ ^	¹¹⁷	¹³³	¹⁴⁹	¹⁶⁵	¹⁸¹ 5	¹⁹⁷ F	²¹³ V	²²⁹ f	²⁴⁵ v	5	0L0L
⁶ SOH	²² CR	³⁸	⁵⁴	⁷⁰	⁸⁶	¹⁰² °	¹¹⁸ @	¹³⁴ ↑	¹⁵⁰	¹⁶⁶ <	¹⁸² 6	¹⁹⁸ G	²¹⁴ W	²³⁰ g	²⁴⁶ w	6	0LLO
⁷ STX	²³ NF	³⁹	⁵⁵	⁷¹	⁸⁷	¹⁰³ ~	¹¹⁹ &	¹³⁵	¹⁵¹ =	¹⁶⁷ >	¹⁸³ 7	¹⁹⁹ H	²¹⁵ X	²³¹ h	²⁴⁷ x	7	0LLL
⁸ ETX	²⁴ VT	⁴⁰	⁵⁶	⁷²	⁸⁸	¹⁰⁴	¹²⁰ *	¹³⁶	¹⁵² ≠	¹⁶⁸ (MZ)}	¹⁸⁴ 8	²⁰⁰ I	²¹⁶ Y	²³² i	²⁴⁸ y	8	L000
⁹ EOT	²⁵ VT3	⁴¹	⁵⁷	⁷³	⁸⁹	¹⁰⁵	¹²¹	¹³⁷	¹⁵³	¹⁶⁹ .	¹⁸⁵ 9	²⁰¹ J	²¹⁷ Z	²³³ j	²⁴⁹ z	9	L00L
¹⁰ ENQ	²⁶ VT4	⁴²	⁵⁸	⁷⁴	⁹⁰	¹⁰⁶	¹²²	¹³⁸	¹⁵⁴	¹⁷⁰ ,	¹⁸⁶	²⁰² K	²¹⁸ Ä	²³⁴ k	²⁵⁰ ä	A	LOLO
¹¹ ACK	²⁷ VT5	⁴³ HT	⁵⁹	⁷⁵	⁹¹	¹⁰⁷ \	¹²³	¹³⁹	¹⁵⁵ <	¹⁷¹ :	¹⁸⁷	²⁰³ L	²¹⁹ Ö	²³⁵ l	²⁵¹ ö	B	L0LL
¹² DLE	²⁸ VT6	⁴⁴ BS	⁶⁰ IS4	⁷⁶	⁹²	¹⁰⁸ ∩	¹²⁴ ◻	¹⁴⁰	¹⁵⁶ >	¹⁷² ;	¹⁸⁸	²⁰⁴ M	²²⁰ Ü	²³⁶ m	²⁵² ü	C	LL00
¹³ NAK	²⁹ VT7	⁴⁵ ESC	⁶¹ IS3	⁷⁷	⁹³	¹⁰⁹)	¹²⁵	¹⁴¹ 10	¹⁵⁷ ≡	¹⁷³ !	¹⁸⁹	²⁰⁵ N	²²¹	²³⁷ n	²⁵³ ß	D	LL0L
¹⁴ SYN	²⁰ VT8	⁴⁶ SO	⁶² IS2	⁷⁸	⁹⁴	¹¹⁰ -	¹²⁶	¹⁴²	¹⁵⁸ ≡	¹⁷⁴ ?	¹⁹⁰	²⁰⁶ O	²²²	²³⁸ o	²⁵⁴	E	LLLO
¹⁵ ETB	³¹ NL7	⁴⁷ SI	⁶³ IS1	⁷⁹	⁹⁵	¹¹¹ -	¹²⁷ π	¹⁴³	¹⁵⁹	¹⁷⁵ SP	¹⁹¹ (PZ)}	²⁰⁷ P	²²³	²³⁹ p	²⁵⁵ DEL	F	LLLL
Steuerzeichen						Akzente	Sonstige Zeichen	Mathematische Zeichen	Satzzeichen	Ziffern	Großbuchstaben	Kleinbuchstaben				GR 310	

3.2. Übertragungssteuerzeichen

Diese Zeichen sind ausschließlich dazu bestimmt, den Betriebsablauf der Übertragung von digitalen Daten zwischen Datenstationen zu steuern.

SOH Anfang des Kopfes (Start of Heading)

Das Zeichen wird am Anfang einer Zeichenfolge verwendet, die eine Adresse und/oder Angaben zur Weiterleitung enthält und maschinell interpretierbar ist.

STX Anfang des Textes (Start of Text)

Das Zeichen leitet eine zusammengehörige Zeichenfolge ein, die vollständig an den Empfänger übermittelt wird. Eine solche Zeichenfolge wird als "Text" bezeichnet. Das Zeichen STX beendet eine Zeichenfolge, die mit SOH beginnt.

ETX Ende des Textes (End of Text)

Das Zeichen beendet eine Zeichenfolge, die mit STX beginnt.

EOT Ende der Übertragung (End of Transmission)

Das Zeichen beendet die Übertragung von einem oder mehreren Texten.

ENQ Stationsaufforderung (Enquiry)

Das Zeichen dient zur Anforderung einer Antwort von einer fernen Station. Die Antwort kann die Stationskennung auf/oder den Stationszustand einschließen.

Wenn in öffentlichen Netzen eine "Wer da?"-Funktion benötigt wird, dann muß das Zeichen ENQ bei seinem ersten Auftreten, nachdem die Verbindung hergestellt ist, "Wer da?" (Aufforderung zur Abgabe der Stationskennung) beinhalten. Je nach Vereinbarung können später auftretende Zeichen ENQ diese Funktion beinhalten oder nicht.

ACK Positive Rückmeldung (Acknowledge)

Das Zeichen ist eine bejahende Antwort der Empfangsstation an die Sendestation.

DLE Datenübertragungsumschaltung (Data Link Escape)

Das Zeichen leitet eine Zeichenfolge ein, die einen außerhalb des Codes liegenden Übertragungssteuerbefehl darstellt. In einer solchen Zeichenfolge dürfen nur Bitkombinationen vorkommen, die in der Code-Tabelle mit Schriftzeichen oder Übertragungssteuerzeichen belegt sind. Die Zeichenfolge und ihre Interpretation bedürfen der Vereinbarung.

NAK Negative Rückmeldung (Negative Acknowledge)

Das Zeichen ist eine verneinende Antwort der Empfangsstation an die Sendestation.

SYN Synchronisierung (Synchronous Idle)

Das Zeichen wird in Synchron-Systemen übertragen, wenn kein anderes Zeichen zum Senden vorliegt, und dient zum Herstellen oder Erhalten des Synchronismus zwischen den Endstellen einer Verbindung.

ETB Ende des Datenübertragungsblocks (End of Transmission Block)

Das Zeichen schließt einen Datenübertragungsblock ab.

3.3. Formatsteuerzeichen (FE)

Diese Zeichen bestimmen die Anordnung der Daten auf Datenträgern für die Ein- und Ausgabe und damit ggf. auch ihre Gliederung. Die nachstehend angegebene Bedeutung der einzelnen Zeichen gilt für zeichenweise druckende Geräte (z.B. Schreibmaschinen und Fernschreibgeräte). Bei anderen Geräten dürfen diese Zeichen mit verwandter Bedeutung benutzt werden; hierzu bedarf es aber besonderer Vereinbarungen.

NF Neues Formular (New Form)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung auf die erste Schreibposition der vorher festgelegten ersten Zeile des nächsten Formulars.

Das Zeichen kann in der Bedeutung von Vorschub gemäß Spur 1 verwendet werden (siehe unter VTx).

NL Neue Zeile (New Line)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung auf die erste Schreibposition der nächsten Zeile.

NLx Vorschub um x Zeilen

Diese Zeichen dienen zur Fortbewegung der Schreibeinrichtung um x Zeilen und zwar auf die erste Schreibposition der angesteuerten Zeile. (Der Vorschub um null Zeilen (NL0) ist durch das Steuerzeichen CR Wagenrücklauf dargestellt, und der Vorschub um eine Zeile (NL1) ist durch das Steuerzeichen NL Neue Zeile dargestellt.)

SP Zwischenraum (Space)

Dieses Zeichen dient zur Vorwärtsbewegung der Schreibeinrichtung um einen Schreibrschritt innerhalb einer Zeile. Es wird Vorwärtsschritt genannt, wenn es nur auf die Steuerfunktion ankommt.

Für dieses Zeichen ist kein abdruckbares Zeichen angegeben. Im Einzelfall kann es zweckmäßig sein, dieses Zeichen z.B. durch \cup oder b (aus englisch "blank") bildlich darzustellen.

BS Rückwärtsschritt (Backspace)

Das Zeichen veranlaßt die Rückbewegung der Schreibeinrichtung um einen Schreibrschritt innerhalb der Zeile.

CR Wagenrücklauf (Carriage Return)

Das Zeichen dient zur Rückbewegung der Schreibeinrichtung auf die erste Schreibposition derselben Zeile.

HT Horizontal-Tabulator (Horizontal Tabulation)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung bis zur nächsten Tabulatorposition innerhalb der Zeile.

VT Vertikal-Tabulator (Vertical Tabulation)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung zu der nächsten, in einer vorher festgelegten Serie von Zeilenpositionen.

Das Zeichen kann in der Bedeutung von Vorschub gemäß Spur 2 verwendet werden (siehe unter VTx).

VTx Vorschub gemäß Spur x

Diese Zeichen dienen zur Fortbewegung der Schreibeinrichtung zu der nächsten in der Spur x vorher festgelegten Serie von Zeilenpositionen.

Die Spur 1 darf nur die Zeilenposition für die erste Zeile des Formulars enthalten. Der Vorschub gemäß Spur 1 (VT1) ist daher durch das Steuerzeichen NF Neues Formular dargestellt. Der Zeilenvorschub gemäß Spur 2 (VT2) wird durch das Steuerzeichen VT dargestellt.

3.4. Gerätesteuerzeichen

Diese Zeichen dienen zum Steuern von Zusatz- und Hilfsgeräten in Datenverarbeitungsanlagen und Übertragungssystemen, insbesondere zum Ein- und Ausschalten dieser Geräte. Beispielsweise können sie bei Lochstreifenbetrieb folgendermaßen benutzt werden:

- DC1 - Erster Locher ein
- DC2 - Zweiter Locher ein
- DC3 - Abtaster ein

DC4 wird zum Abschalten der Geräte verwendet.

Das Benutzen der Gerätesteuerzeichen bedarf besonderer Vereinbarungen.

3.5. Informationstrennzeichen

Die vier Trennzeichen

- IS1 Teilgruppen-Trennung (US)
- IS2 Untergruppen-Trennung (RS)
- IS3 Gruppen-Trennung (GS) und
- IS4 Hauptgruppen-Trennung (FS)

dienen zur logischen Gliederung von Daten. Sie werden in der vorstehenden Rangfolge verwendet, wobei die Teilgruppe die kleinste Einheit ist.

3.6. Steuerzeichen zur Code-ErweiterungESC Umschaltung (Escape)

Das Zeichen verändert die Bedeutung des nächstfolgenden Zeichens gegenüber der in der Code-Tabelle angegebenen. Falls erforderlich, kann dieses nächstfolgende Zeichen wiederum eine Zeichenfolge einleiten, die als außerhalb des Codes liegend zu interpretieren ist. Die durch Umschaltung eingeleitete Zeichenfolge (das einzelne Zeichen und die ggf. davon abhängige Zeichenfolge) soll die Zeichen Nil, Löschen und die zehn Übertragungssteuerzeichen nicht enthalten, es sei denn in ihrer ursprünglichen Bedeutung.

Eine durch Umschaltung eingeleitete Zeichenfolge stellt einen außerhalb des Codes liegenden Steuerbefehl dar, jedoch keinen Übertragungssteuerbefehl. Solche Steuerbefehle können u.a. dazu dienen, zusätzliche Schriftzeichen und Alphabete zu gewinnen. Das auf Umschaltung folgende Zeichen, die ggf. davon abhängige Zeichenfolge und beider Interpretation bedürfen der Vereinbarung.

SO Dauerumschaltung (Shift-out)

Die auf das Zeichen Dauerumschaltung folgenden Zeichen sind nicht nach der Code-Tabelle zu interpretieren - mit Ausnahme der Steuerzeichen (Spalten 0 und 1 und Platz 7/15). Eine durch Dauerumschaltung und Rückschaltung begrenzte Zeichenfolge dient ausschließlich dazu, zusätzliche Schriftzeichen zu gewinnen. Die Zeichenfolge und ihre Interpretation bedürfen der Vereinbarung.

SI Rückschaltung (Shift-in)

Die auf das Zeichen Rückschaltung folgenden Zeichen werden nach der Code-Tabelle interpretiert.

3.7. Sonstige SteuerzeichenNUL Nil (Null)

Dieses Zeichen ist ein Füllzeichen. Es kann also einer Zeichenfolge hinzugefügt oder in dieser unterdrückt werden, ohne daß sich ihre Bedeutung ändert.

NU x Nil x

Eines dieser Zeichen kann am Ende eines Satzes in Dateien stehen. Es zeigt an, daß x bedeutungslose Zeichen bis zur Wortgrenze hinzugefügt wurden.

BEL Klingel (Bell)

Das Zeichen bewirkt ein vorzugsweise akustisches Signal und kann auch Anzeigergeräte auslösen.

CAN Ungültig (Cancel)

Das Zeichen bedeutet, daß die vorangehenden Zeichen Fehler enthalten oder eliminiert werden sollen. Es ist besonders zu vereinbaren, welcher Abschnitt zu berichtigen ist. Er soll vorzugsweise durch ein Steuerzeichen (z.B. ein Informationstrennzeichen) begrenzt sein.

FL Fluchtsymbol

Dieses Zeichen leitet eine Zeichenfolge ein, die nach der Syntax der Kommando-Sprache zu interpretieren ist.

EM Ende der Aufzeichnung (End of Medium)

Dieses Zeichen kann benutzt werden, um das Ende des gewünschten oder verwendeten Teils der Aufzeichnung auf einem Datenträger einzugeben.

SUB Substitution (Substitute Character)

Dieses Zeichen dient dazu, ein Zeichen zu ersetzen, das als nicht zulässig oder fehlerhaft erkannt worden ist.

DEL Löschen (Delete)

Das Zeichen wird vorzugsweise zum Auslöschen oder Überschreiben fehlerhafter oder nicht gewünschter Zeichen in Lochstreifen benutzt. Es kann auch als Füllzeichen dienen, also einer Zeichenfolge hinzugefügt oder in dieser unterdrückt werden, ohne daß sich ihre Bedeutung ändert; beeinflusst wird nur die Darstellung der Informationen und/oder die Gerätesteuerung.

MZ Minus Null

Steht für -0, wenn dies durch ein Zeichen dargestellt werden soll.

PZ Plus Null

Steht für +0, wenn dies durch ein Zeichen dargestellt werden soll.

TE Textende

Dieses Zeichen zeigt das Ende einer Zeichenfolge (eines Textes) an.

4. Benennung der Sonderzeichen

Dezimal Sede- Zeichen Benennung
 zimal

96	'60'	"	Anführungszeichen
97	'61'	'	Apostroph
98	'62'	´	Akut
99	'63'	˘	Gravis
101	'65'	ˆ	Zirkumflex
102	'66'	˚	Ringel
103	'67'	˜	Tilde
107	'6B'	\	Inverser Schrägstrich
108	'6C'	ƒ	Zeichenreihenklammer auf
109	'6D'	ƒ	Zeichenreihenklammer zu
110	'6E'	—	Unterstreichung
111	'6F'	—	Überstreichung
112	'70'	%	Prozent
113	'71'	§	Paragraph
114	'72'	#	Nummer
115	'73'	\$	Dollar
116	'74'	¢	Cent
118	'76'	@	kommerzielles @
119	'77'	&(e)	kommerzielles Und
120	'78'	*	Stern
124	'7C'	☒	Kissen
127	'7F'	π	Pi
128	'80'	^	Konjunktion
129	'81'	v	Disjunktion
130	'82'	¬	Negativ

134	'86'	†	Pfeil nach oben
139	'8B'		senkrechter Strich
141	'8D'	∩	Basiszehn
144	'90'	+	plus
145	'91'	-	minus
147	'93'	/	Schrägstrich
151	'97'	=	gleich
152	'98'	≠	nicht gleich
155	'9B'	<	kleiner als
156	'9C'	>	größer als
157	'9D'	≤	kleiner oder gleich
158	'9E'	≥	größer oder gleich
160	'A0'	(runde Klammer auf
161	'A1')	runde Klammer zu
162	'A2'	[eckige Klammer auf
163	'A3']	eckige Klammer zu
164	'A4'	{	geschweifte Klammer auf
165	'A5'	}	geschweifte Klammer zu
166	'A6'	<	spitze Klammer auf
167	'A7'	>	spitze Klammer zu
169	'A9'	.	Punkt
170	'AA'	,	Komma
171	'AB'	:	Doppelpunkt
172	'AC'	;	Semikolon
173	'AD'	!	Ausrufezeichen
174	'AE'	?	Fragezeichen
175	'AF'		Zwischenraum

5. Bedeutung und Form der Schriftzeichen

Die Bedeutung der Schriftzeichen wird hier nicht festgelegt. Sie muß je nach Anwendung vereinbart werden, sofern in einschlägigen Normen, z.B. für Programmiersprachen, nichts darüber ausgesagt ist. Zu vermeiden ist jedoch jede Interpretation, die im Gegensatz zu einer gebräuchlichen Bedeutung steht. Die Benutzung eines Schriftzeichens in mehreren Bedeutungen ist zulässig, z.B. kann das Zeichen *minus* auch Bindestrich oder Gedankenstrich oder Trennungsstrich bedeuten.

Die Form der Schriftzeichen ist hier nicht im einzelnen festgelegt.



6. Gebräuchliche Bedeutung der Sonderzeichen

Für einige Sonderzeichen wird hier die gebräuchlichste Bedeutung aufgeführt:

- 108 Zeichenreihenklammern bei ALGOL
- 109 dto.
- 114 Kennzeichnung einer Nummer zwischen Zahlen
- 118 im Sinne von je, pro, à
- 120 Malzeichen bei den problemorientierten Sprachen
- 124 bedeutsame Leerstelle (z.B. ⌘ 120.-- DM)
- 128 Konjunktion; UND-Verknüpfung
- 129 Adjunktion (Disjunktion); ODER-Verknüpfung
- 134 potenzieren
- 139 oder
- 147 Division bei den problemorientierten Sprachen

Erläuterungen

Im Zentralcode sind alle Zeichen und Steuerzeichen aufgeführt, soweit ein Bedarf bestand.

Die früher bereits belegten Plätze sind zum Teil für diese Bedeutung reserviert worden, liegen aber damit keinesfalls fest.

Wird ein Zeichen oder Steuerzeichen benötigt, das nicht in dieser Werknorm vorhanden ist, so ist in Zusammenarbeit mit der Arbeitsgruppe AG3, Codes, diese Werknorm zu ergänzen.

Soll ein hier festgelegtes Zeichen oder Steuerzeichen in einer hier nicht festgelegten, aber ähnlichen Bedeutung verwendet werden, so ist dies mit der Arbeitsgruppe AG3, Codes, zu vereinbaren.

Die Bedeutung der Steuerzeichen in Bezug auf ein bestimmtes Ein- oder Ausgabegerät ist der jeweiligen Werknorm über den Ein- bzw. Ausgabecode zu entnehmen.

Gegenüber der Ausgabe von November 69 wurden zusätzlich alle die Steuerzeichen des internationalen 7-Bit-Codes nach DIN 66 003 aufgenommen, die noch nicht im ZC1 enthalten waren.

Der internationale 7-Bit-Code gewinnt immer mehr an Bedeutung. Aus diesem Grunde sei dem Anwender empfohlen, nur die Zeichen des ZC1 zu verwenden, die im 7-Bit-Code nach DIN 66 003 (Werknorm 2N 0812.411) vorhanden sind, und zwar die nicht eingeklammerten Zeichen. Bedingt durch den z.Z. in Arbeit befindlichen internationalen 8-Bit-Code könnten evtl. bei anderen Zeichen Schwierigkeiten auftreten.

In diesem neuen Entwurf sind wegen der Anpassung an den 7-Bit-Code nach DIN 66 003 die geschweiften Klammern auf die Plätze 168 bzw. 191 verlegt worden.



Informationsverarbeitung
Zeichensätze

ZS

2N 0810

.111

Ein Zeichensatz ist ein vereinbarter Vorrat von Schriftzeichen. Für den TR 440 werden Zeichensätze festgelegt mit

- 48 Zeichen (ZS 48)
- 54 Zeichen (ZS 54)
- 61 Zeichen (ZS 61)
- 64 Zeichen (ZS 64)
- 115 Zeichen (ZS 115)
- 120 Zeichen (ZS 120)

Hierbei ist der kleinere Zeichensatz jeweils im größeren enthalten. Auf diese Zeichensätze wird Bezug genommen, wenn für textverarbeitende Programme Textdarstellungen festzulegen sind.

Bei der Festlegung der Zeichensätze wurden folgende Punkte berücksichtigt:

- Zeichensätze, die für die problemorientierten Programmiersprachen festliegen,
- Zeichensätze, die für Rechnerendgeräte festliegen,
- Begrenzung der Zeichenmenge bei den Rechnerendgeräten.

1. Zeichensatztablelle

(Tabelle Seite 2)

In der Zeichensatztablelle sind die Zeichen der verschiedenen Zeichensätze aufgelistet. Die Reihenfolge der Zeichen und ihre fortlaufende Nummer ist nur für die hier getroffene Festlegung der Zeichensätze gewählt und drückt darüber hinaus keine Sortierfolge aus. Die Benennung ist vorzugsweise nach der Form des Zeichens und ggf. nach der wichtigsten Bedeutung dieses Zeichens gewählt.

Die Bedeutung der Zeichen wird hier nicht festgelegt. Sie muß je nach Anwendung vereinbart werden, sofern in einschlägigen Normen, z.B. für Programmiersprachen, nichts darüber gesagt ist. Zu vermeiden ist jedoch jede Interpretation, die im Gegensatz zu einer gebräuchlichen Bedeutung steht. Die Benutzung eines Schriftzeichens in mehreren Bedeutungen ist zulässig, z.B. kann das Zeichen "minus" auch Bindestrich oder Gedankenstrich oder Trennungsstrich bedeuten.

Die Form der Zeichen wird im einzelnen hier ebenfalls nicht festgelegt.

Jedem Zeichen ist der Bindrwert angefügt, der ihm im Zentralcode des TR 440 zugeordnet ist. Er ist als Dezimal- und Sedezimalzahl angegeben.

2. Spezielle Zeichen

Das Ausrufezeichen (Lfd. Nummer 48) dient bei einigen Umcodierungen nicht als Schriftzeichen, sondern zur Code-Erweiterung.

Das Zeichen Kissen (Lfd. Nummer 54) wird auch als Steuerzeichen zur Steuerung des Rechners (Fluchtsymbol für Kommandos) verwendet.

3. Zeichensatz ZS 48

Der Zeichensatz ZS 48 ist auf Geräte abgestimmt, die nur eine Zeichenmenge von 48 Zeichen besitzen. Es ist dies der Kartenlocher 26 der Firma IBM. Gegenüber der Symbolausführung H wurde an Stelle des Dollarzeichens (\$) das Ausrufezeichen (!) verwendet.

4. Zeichensatz ZS 54

Der Zeichensatz ZS 54 ist auf den Fernschreiber mit maximal 54 Schriftzeichen abgestimmt. Gegenüber DIN 66 006, Darstellung von ALGOL-Symbolen auf 5-Spur-Lochstreifen, die auf dem Internationalen Telegrafien-Alphabet Nr. 2 (CCITT) basiert, bestehen folgende Abweichungen:

ZS 54	DIN 66 006	CCITT
*	x	?
;	;	∞
[[
]]	
!	10	
&		
⊞		

5. Zeichensatz ZS 61

Der Zeichensatz ZS 61 stimmt mit dem Zeichensatz des Sichtgerätes (Code SIC 1) überein.

6. Zeichensatz ZS 64

Der Zeichensatz ZS 64 ist abgestimmt auf Kartenlocher mit 64 Zeichen und auf Schnelldrucker. Der Zeichensatz ZS 64 ist identisch mit dem Zeichenvorrat des Schnelldruckers (Code DC1).

Gegenüber dem Kartenlocher 29 der Fa. IBM bestehen zwischen der USA-Version und der deutschen Version der Symbolausführung EL folgende Unterschiede:

Symbolausführung EL - USA	#	@	\$
Symbolausführung EL - deutsch	Ä	Ö	Ü

Außerdem weicht der Zeichensatz 64 von den folgenden, in der amerikanischen und deutschen Symbolausführung EL vorhandenen Zeichen ab:

Symbolausführung EL USA/deutsch	"	⊞	Leerstelle
Zeichensatz ZS 64	[]	⊞

Die eckigen Klammern sind für die ALGOL-Benutzer vorgesehen.

Ähere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:

Rm

N3/NM

GR/V23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2

V

7. Zeichensatz ZS 115

Der Zeichensatz ZS 115 stimmt mit dem Zeichensatz des Schnelldruckers mit 115 Schriftzeichen (Code DC 2) überein.

8. Zeichensatz ZS 120

Der Zeichensatz ZS 120 enthält über ZS 115 hinaus noch 4 bei ALGOL verwendete Zeichen (\leq , \geq , \neq , \dagger) sowie das Zeichen Cent (¢). Diese Zeichen können durch Doppeldruck realisiert werden.

ZEICHENSATZ - TABELLE

ZC 1				
Lfd.Nr.	Benennung	Zeichen	Dezim.	Sedez.
1	Ziffer	0	176	'BO'
⋮	⋮	⋮	⋮	⋮
10	Ziffer	9	185	'B9'
11	Großbuchstabe	A	192	'CO'
⋮	⋮	⋮	⋮	⋮
36	Großbuchstabe	Z	217	'D9'
37	Zwischenraum		175	'AF'
38	plus	+	144	'90'
39	minus	-	145	'91'
40	Stern	*	120	'78'
41	Schrägstrich	/	147	'93'
42	gleich	=	151	'97'
43	runde Klammer auf	(160	'AO'
44	runde Klammer zu)	161	'A1'
45	Punkt	.	169	'A9'
46	Komma	,	170	'AA'
47	Apostroph	'	97	'61'
48	Ausrufezeichen ¹⁾	!	173	'AD'
49	eckige Klammer auf	[162	'A2'
50	eckige Klammer zu]	163	'A3'
51	Doppelpunkt	:	171	'AB'
52	Semikolon	;	172	'AC'
53	kommerzielles Und	&(&)	119	'77'
54	Kissen ²⁾	☐	124	'7C'
55	Prozent	%	112	'70'
56	kleiner als	<	155	'9B'
57	größer als	>	156	'9C'
58	Unterstreichung	—	110	'6E'
59	Negation	¬	130	'82'
60	senkrechter Strich		139	'8B'
61	Fragezeichen	?	174	'AE'
62	Umlaut groß	Ä	218	'DA'
63	Umlaut groß	Ö	219	'DB'
64	Umlaut groß	Ü	220	'DC'

ZC 1				
Lfd.Nr.	Benennung	Zeichen	Dezim.	Sedez.
65	Kleintuchstabe	a	224	'FO'
⋮	⋮	⋮	⋮	⋮
90	Kleinbuchstabe	z	249	'F9'
91	Umlaut klein	ä	250	'FA'
92	Umlaut klein	ö	251	'FB'
93	Umlaut klein	ü	252	'FC'
94	Dollar	\$	115	'73'
95	kommerzielles a	@	118	'76'
96	Kommazeichen	#	114	'72'
97	Anführungszeichen	"	96	'60'
98	Zeichenklammer auf	'	108	'6C'
99	Zeichenklammer zu	`	109	'6D'
100	Konjunktion	^	128	'80'
101	Disjunktion	v	129	'81'
102	Basiszehn	10	141	'8D'
103	geschw. Klammer auf	{	164	'A4'
104	geschw. Klammer zu	}	165	'A5'
105	Paragraph	§	113	'71'
106	Pi	π	127	'7F'
107	Gravis	˘	99	'63'
108	Zirkumflex	^	101	'65'
109	Akut	˙	98	'62'
110	Tilde	~	103	'67'
111	Überstreichung	—	111	'6F'
112	Ringel (over dot)	◦	102	'66'
113	Eszet	ß	253	'FD'
114	spitze Klammer auf	<	166	'A6'
115	spitze Klammer zu	>	167	'A7'
116	kleiner oder gleich	≤	157	'9D'
117	größer oder gleich	≥	158	'9E'
118	ungleich	≠	152	'98'
119	Pfeil nach oben	†	134	'86'
120	Cent	¢	116	'74'

¹⁾ Bei einigen Umcodierungen Steuerzeichen zur Code-Erweiterung (s. Abschnitt 2.)

²⁾ Wird auch als Steuerzeichen für die Kommandosprache verwendet, -Kommando-Fluchtsymbol- (s. Abschnitt 2.)



Informationsverarbeitung
Umcodierleistung Lochkarteneing. TR 440
 Binär

2N 0815

.110

Einsprüche bis 31. März 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3; Codes, Schriftführer: Böckmann N3/V23, Tel. 474

1. Zweck und Anwendung

Diese Werknorm legt die Zuordnung eines Lochmusters auf einer Lochkarte zu einem Bitmuster im TR 440 fest (Binäreingabe).

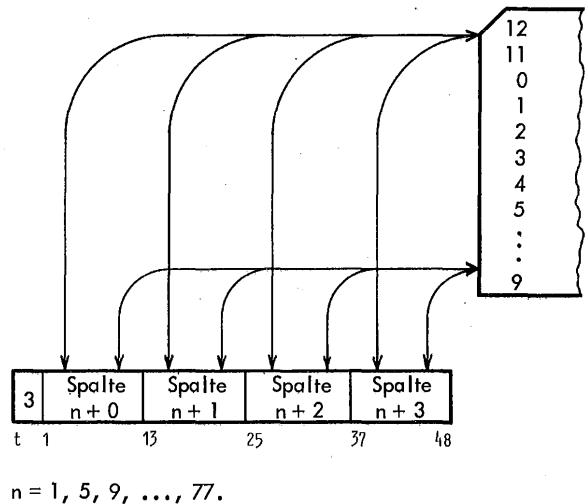
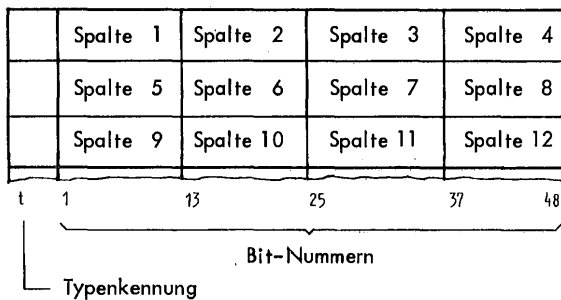
Lochkartenzeile	Bits eines Wortes			
12	1	13	25	37
11	2	14	26	38
0	3	15	27	39
1	4	16	28	40
2	5	17	29	41
3	6	18	30	42
4	7	19	31	43
5	8	20	32	44
6	9	21	33	45
7	10	22	34	46
8	11	23	35	47
9	12	24	36	48

2. Bezeichnung

Die Lochkartenzeilen sind von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet. Die Bits des TR 440-Wortes sind mit den Zahlen 1 bis 48 bezeichnet, wobei das Bit 1 die Wertigkeit 2^{47} und das Bit 48 die Wertigkeit 2^0 hat. t bezeichnet die Typenkennung.

3. Zuordnung

Jeweils 4 Spalten einer Lochkarte werden in einem Ganzwort abgelegt. Pro Karte entstehen also 20 Ganzwörter. Pro Spalte wird ein Viertelwort (12 Bits) benötigt. Die Zuordnung von Spalten zu Wörtern zeigt die nachstehende Abbildung.



Die Typenkennung wird auf den Wert 3 gesetzt.

Die Zuordnung der Lochungen einer Spalte zum Bitmuster eines Wortes zeigt die nachstehende Abbildung.

Ein Loch in der Karte bewirkt, daß das zugehörige Bit auf den Wert L gesetzt wird.

N3/NM

N31/V23

Böckmann

Ersatz für:

Ersetzt durch:

V

Frühere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:



Informationsverarbeitung
Lochkartencode

2N 0813

KC1

.111

1. Zweck und Anwendung

Der Lochkartencode KC1 enthält alle Zeichen des Zeichensatzes ZS64 (Werknorm 2N 0810.111).

Alle Zeichen dieses Codes sind auch im Druckercode DC1 und DC2 enthalten. (Werknormen 2N 0813.311 und 312).

2. Code-Tabelle

Die Lochkartenzeilen werden von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet. Jedes darzustellende Zeichen wird in einer Spalte senkrecht zur Zeile gelocht.

In der Tabelle ist angegeben, welche Stellen einer Spalte gelocht sein müssen, um das Zeichen darzustellen. Die am Kopf der Tabelle und links der Tabelle angegebenen Stellen ergeben das Lochmuster für das Zeichen. Dabei bedeutet das Zeichen "-" keine Lochung.

Code-Tabelle

Lochung	12	11	0	-
-	&	-	0	SP
1	A	J	/	1
2	B	K	S	2
3	C	L	T	3
4	D	M	U	4
5	E	N	V	5
6	F	O	W	6
7	G	P	X	7
8	H	Q	Y	8
9	I	R	Z	9
2-8] [! "	¤	:
3-8	.	Ü	,	Ä
4-8	<	*	%	Ö
5-8	()	-	'	
6-8	+ ;	>	=	
7-8		¬	?	[

Erläuterungen

Die Lochungen entsprechen denen des Kartenlochers 29 der IBM.

Ausgehend von der Symbolausführung EL des Kartenlochers 29 der IBM wurden folgende Zeichen geändert:

IBM EL	KC1
#	ˆ
@	ö
\$	ü
"	[]
¢	¤
Leerstelle	∩

Damit wurde den Forderungen der kaufmännischen Anwendung nach den Umlauten entsprochen, die in einer deutschen Version der IBM EL ebenfalls an der gleichen Stelle vorgesehen sind.

Die eckigen Klammern sind für den ALGOL-Benutzer von Bedeutung.

Ärhere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:

GR 178

N3/NM

N31/V23

Ersatz für:

Ersetzt durch:

V



Informationsverarbeitung
Umcodierleistung Lochkarteneingabe TR 440
 KC1 → ZC1

2N 0815

.111

1. Zweck und Anwendung

Der Karteneingabecode KEC1 stellt eine Zuordnung von Lochkombinationen zu Bitmustern von 8 Bits dar.

Durch den Karteneingabecode KEC1 wird, ausgehend vom Kartencode KC1 (Werknorm 2N 0813.111) eine Zuordnung gemäß Zentralcode ZC1 (Werknorm 2N 0812.511) erreicht.

Der vorliegende Karteneingabecode hat folgende Kennzeichnungen:

symbolischer Name bei der Steuerung der Eingabeprozedur: KC1

Kennzeichen für Codeeinstellung (siehe Abschnitt 4.1. und 5.1.): 1

bezieht sich auf Lochkartencode (Werknorm 2N 0813.111): KC1

2. Bezeichnung

Für die Darstellung des Lochmusters einer Spalte werden die Zeilen der Lochkarte von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet.

In der Code-Tabelle ist jeweils oben im Feld das Zeichen angegeben, das der Lochkombination gemäß dem Kartencode KC1 (Werknorm 2N 0813.111) zugeordnet ist. Im unteren Teil des Feldes ist der dezimale Wert der Oktade angegeben, der diese Lochkombination zugeordnet wird.

3. Zuordnung

Die Zuordnung ist der Code-Tabelle zu entnehmen. Den Lochkombinationen, denen in der Code-Tabelle kein Dezimalwert zugeordnet ist, wird die Oktade mit dem Dezimalwert 32 zugeordnet. Außer der in der Code-Tabelle angegebenen, werden folgende Zuordnungen gemacht:

	Oktade	
	Symbol	dezimal
Kartenende	NL	21
Abschnittsende	EM	33
Keine Zuordnung in a. Code-Tabelle angegeben	SUB	32

4. Fluchtsymbol

4.1. Codeunabhängiges Fluchtsymbol

Das codeunabhängige Fluchtsymbol wird durch die Lochung

12-11-5-8 in der Spalte 1

dargestellt. Es wird stets unmittelbar von dem Kennzeichen für die Codeeinstellung (siehe Abschnitt 1) gefolgt. Es wird nur zur Einleitung der Kommandos für die Eingabeprozedur verwendet (siehe Abschnitt 5.1.).

4.2. Codeabhängiges Fluchtsymbol

Das codeabhängige Fluchtsymbol ist für den vorliegenden Code die Lochung

0-2-8 in der Spalte 1

Ihr ist im Kartencode KC1 das Zeichen \square zugeordnet. In dem in Abschnitt 5.2. angegebenen Fall darf das codeabhängige Fluchtsymbol in jeder Spalte stehen.

Des weiteren kann durch die Eingabeprozedur festgelegt werden, daß das codeabhängige Fluchtsymbol in jeder Spalte zugelassen ist.

5. Spezielle Folgen

In den nachstehenden Fällen wird eine andere Zuordnung von Lochmustern zu Oktaden vorgenommen, als in Abschnitt 3. angegeben.

5.1. Anweisungen an die Eingabeprozedur

Folgt unmittelbar auf ein codeunabhängiges Fluchtsymbol mit nachfolgendem Kennzeichen für die Codeeinstellung oder - soweit dies in der Eingabeprozedur vorgesehen ist - auf ein codeabhängiges Fluchtsymbol das Zeichen

X (Lochung 0-7)

so ist dies eine Anweisung an die Eingabeprozedur. Ein codeabhängiges Fluchtsymbol mit nachfolgendem Punkt beendet die Folge; in diesem Fall ist das codeabhängige Fluchtsymbol in jeder Spalte zugelassen.

Die Zuordnung der Lochungen der vorstehenden Anweisungen zu Oktaden wird durch die Eingabeprozedur festgelegt.

Diese Unterlage darf weder kopiert, noch
 Dritten Personen mitgeteilt, noch ander-
 weitig mißbräuchlich benutzt werden

erungen:

Rm

5.2. Dezimaler Oktadenwert

Falls die Darstellung einer beliebigen Oktade durch ihren dezimalen Wert in der Eingabeprozedur vorgesehen ist, gilt nachstehende Festlegung.

Folgen einem codeabhängigen Fluchtsymbol drei Ziffern (zugehörige Lochungen siehe Code-Tabelle), so stellt diese dreistellige Dezimalzahl einen Oktadenwert dar. Der Folge (Fluchtsymbol, drei Ziffern) wird der angegebene Oktadenwert zugeordnet.

{ 0 9 6 } ergibt { 96 }

Es sind nur die Werte von 064 bis 255 zugelassen.

Alle 4 Zeichen müssen in derselben Lochkarte sein.

Erläuterungen

Durch diese Norm wird die Steuerung der Eingabeprozedur (siehe Abschnitt 5.1.) nicht festgelegt.

Auch die Möglichkeit, daß durch das Kommando die Eingabeprozedur so umgesteuert werden kann, daß eine andere als die hier festgelegte Zuordnung erreicht werden kann und auch, daß auf einen anderen Code umgeschaltet werden kann, wurde hier nicht angegeben.

Code-Tabelle

Lochung	12	11	0	-	12-0	12-11	11-0	12-11-0
-	& 119	- 145	0 176	SP 175	PZ 191		MZ 168	
1	A 192	J 201	/ 147	1 177				
2	B 193	K 202	S 210	2 178				
3	C 194	L 203	T 211	3 179				
4	D 195	M 204	U 212	4 180				
5	E 196	N 205	V 213	5 181				
6	F 197	O 206	W 214	6 182				
7	G 198	P 207	X 215	7 183				
8	H 199	Q 208	Y 216	8 184				
9	I 200	R 209	Z 217	9 185				
2-8] 163	I 173	{ 53	: 171				
3-8	. 169	Ü 220	, 170	Ä 218				
4-8	< 155	* 120	% 112	Ö 219				
5-8	(160) 161	170	' 97				
6-8	+ 144	; 172	> 156	= 151				
7-8	139	¬ 130	? 174	[162				

GR 175



Informationsverarbeitung
Umcodierleistung Lochkartenausg. TR 440
 ZC1 → KC1

2N 0815

.211

Einsprüche bis 31. März 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3: Codes, Schriftführer: Böckmann N31/V23, Tel 474

1. Zweck und Anwendung

Diese Werknorm legt fest, welche Umcodierleistung bei der Lochkartenausgabe des TR 440 im Code KC1 erbracht wird. Den Oktaden des Zentralcodes ZC1 (Werknorm 2N 0812.511) werden Lochmuster der Lochkartencodes KC1 (Werknorm 2N 0813.111) zugeordnet.

2. Bezeichnung

In der Code-Tabelle wird vom Bitmuster des Zentralcodes ZC1 ausgegangen. In der Kopfleiste und der rechten Seitenleiste sind Dualwert und Sedezimalwerte des ZC1 und in der linken oberen Ecke eines jeden Feldes der Dezimalwert der ZC1-Oktade angegeben. Die weiter im Feld vorhandene Angabe bezeichnet das Lochmuster, das der ZC1-Oktade zugeordnet wird. Die Lochkartenzeilen sind von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet.

Die Übersichtstabelle entspricht dem Zentralcode ZC1 und gibt eine Übersicht über die Zeichen, die auf dem Drucker gemäß dieser Werknorm dargestellt werden können. Sie ist nicht Bestandteil dieser Werknorm.

3. Informationsdarstellung

Die auszugebende Information hat die Form einer Datei vom Typ sequentiell. Jeder Satz der Datei bildet eine Karte. Die Information der Datei kann im A-Format oder im O-Format vorliegen.

3.1. A-Format

Das erste Zeichen jedes Satzes wird ignoriert.

3.2. O-Format

Das erste Zeichen jedes Satzes bildet das erste Zeichen der Lochkarte.

3.3. Länge eines Satzes

Enthält ein Satz weniger als 80 Zeichen, so bleibt der Rest der Karte leer (auffüllen mit SP Zwischenraum).

Enthält ein Satz mehr als 80 Zeichen, so werden die überzähligen Zeichen so behandelt, als wären sie nicht vorhanden.

4. Zuordnung

Die Zeichen des Zentralcodes werden gemäß der Code-Tabelle dem Lochmuster zugeordnet. Für Zeichen, deren Felder punktiert sind, wird ein Ersatzzeichen ausgegeben. Für das Zeichen Space (dez. 175) wird keine Lochung

ausgegeben. Für alle Zeichen, denen in der Code-Tabelle keine Lochung zugeordnet ist, wird das Lochmuster 12-11-1-2-3-4-5-6-7-8-9 (vollständig durchlochte Spalte) ausgegeben.

ZC1			KC1	
Zeichen	dez.		Zeichen	Lochg.
FL Fluchtsymbol	53	Y	codeunabhg. Fluchtsymbol	5-8-11-12
NUL Nil	0		keine Zuordnung	
NUL	1		keine Zuordnung	
:	:		keine Zuordnung	
NU5	5		keine Zuordnung	
10 Basiszehn	141	'	Apostroph	5-8
alle nicht angegebenen ZC1-Zeichen			Durchlochung	
Kleinbuchstaben			Großbuchstaben *)	

5. Sonstiges

Bei der Ausgabe werden in der nachstehenden Reihenfolge ausgegeben:

- 1 Leerkarte
- 1 Klarschriftkarte
- Informationskarten
- 1 Klarschriftendekarte

Durch Aussteuern einer Leerkarte werden Abschnitte bzw. Dateien optisch getrennt.

5.1. Leerkarte

Eine Leerkarte enthält keine Lochung. (Alle Spalten erhalten SP Zwischenraum.)

5.2. Klarschriftkarte

Die Klarschriftkarte enthält ein Lochmuster, das Klarschrift darstellt und vom Bedienungspersonal für die Identifizierung gelesen werden kann. Die Klarschrift beginnt ab Spalte 5. Die Klarschriftendekarte enthält darüber hinaus eine Durchlochung in den Spalten 79 und 80.

*) Es steht noch nicht fest, ob den Kleinbuchstaben Großbuchstaben oder eine Durchlochung zugeordnet wird.

Frühere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:


Übersichts - Tabelle

ZC1→KC1

2⁷ 2⁰
1. Sede-
zimale 2. Sede-
zimale

0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	L0LL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0 NUL	16 NL6	32 SUB	48	64	80	96 "	112 %	128 ^	144 +	160 (176 0	192 A	208 Q	224 a	240 q	0	0000
1 NU1	17 NL5	33 EM	49	65	81	97 ,	113 §	129 v	145 -	161)	177 1	193 B	209 R	225 b	241 r	1	000L
2 NU2	18 NL4	34	50	66	82	98 /	114 #	130 ¬	146	162 [178 2	194 C	210 S	226 c	242 s	2	00LO
3 NU3	19 NL3	35	51	67	83	99 ,	115 \$	131 /	147	163]	179 3	195 D	211 T	227 d	243 t	3	00LL
4 NU4	20 NL2	36	52	68	84	100	116 ¢	132	148	164 {	180 4	196 E	212 U	228 e	244 u	4	0LOO
5 NU5	21 NL	37 TE	53 FL	69	85	101 ^	117	133	149	165 }	181 5	197 F	213 V	229 f	245 v	5	0LOL
6	22 CR	38	54	70	86	102 o	118 @	134 ↑	150	166 <	182 6	198 G	214 W	230 g	246 w	6	0LLO
7	23 NF	39	55	71	87	103 ~	119 &	135	151 =	167 >	183 7	199 H	215 X	231 h	247 x	7	0LLL
8	24 VT	40	56	72	88	104 ,	120 *	136	152 +	168 MZ	184 8	200 I	216 Y	232 i	248 y	8	L000
9	25 VT3	41	57	73	89	105	121	137	153	169 .	185 9	201 J	217 Z	233 j	249 z	9	L00L
10	26 VT4	42	58	74	90	106	122	138	154	170 ,	186	202 K	218 Ä	234 k	250 ä	A	LOLO
11	27 VT5	43	59	75	91	107 /	123	139	155 <	171 :	187	203 L	219 Ö	235 l	251 ö	B	LOLL
12	28 VT6	44	60	76	92	108 /	124 □	140	156 >	172 ;	188	204 M	220 Ü	236 m	252 ü	C	LLOO
13	29 VT7	45	61	77	93	109 ,	125	141 10	157 ≡	173 !	189	205 N	221	237 n	253 ñ	D	LLOL
14	30 VT8	46	62	78	94	110 -	126	142	158 ≡	174 ?	190	206 O	222	238 o	254	E	LLLO
15	31 NL7	47	63	79	95	111 -	127 π	143	159	175 SP	191 PZ	207 P	223	239 p	255	F	LLLL

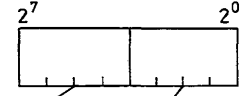
 wird durch Ersatzzeichen dargestellt

 nicht erlaubte Zeichen

GR 2a

Code - Tabelle

ZC1 → KC1



0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	LOLL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0	16	32	48	64	80	96	112 0-4-8	128	144 12-6-8	160 12-5-8	176 0	192 12-1	208 11-8	224 12-1	240 11-8	0	0000
1	17	33	49	65	81	97 5-8	113	129	145 11	161 11-5-8	177 1	193 12-2	209 11-9	225 12-2	241 11-9	1	000L
2	18	34	50	66	82	98	114	130 11-7-8	146	162 7-8	178 2	194 12-3	210 0-2	226 12-3	242 0-2	2	00LO
3	19	35	51	67	83	99	115	131	147 0-1	163 12-2-8	179 3	195 12-4	211 0-3	227 12-4	243 0-3	3	00LL
4	20	36	52	68	84	100	116	132	148	164	180 4	196 12-5	212 0-4	228 12-5	244 0-4	4	0LOO
5	21	37	53 5-8-11-12	69	85	101	117	133	149	165	181 5	197 12-6	213 0-5	229 12-6	245 0-5	5	0LOL
6	22	38	54	70	86	102	118	134	150	166	182 6	198 12-7	214 0-6	230 12-7	246 0-6	6	0LLO
7	23	39	55	71	87	103	119 12	135	151 6-8	167	183 7	199 12-8	215 0-7	231 12-8	247 0-7	7	0LLL
8	24	40	56	72	88	104	120 11-4-8	136	152	168 11-0	184 8	200 12-9	216 0-8	232 12-9	248 0-8	8	L000
9	25	41	57	73	89	105	121	137	153	169 12-3-8	185 9	201 11-1	217 0-9	233 11-1	249 0-9	9	L00L
10	26	42	58	74	90	106	122	138	154	170 0-3-8	186	202 11-2	218 3-8	234 11-2	250 3-8	A	LOLO
11	27	43	59	75	91	107	123	139 12-7-8	155 12-4-8	171 2-8	187	203 11-3	219 4-8	235 11-3	251 4-8	B	L0LL
12	28	44	60	76	92	108	124 0-2-8	140	156 0-6-8	172 11-6-8	188	204 11-4	220 11-3-8	236 11-4	252 11-3-8	C	LL00
13	29	45	61	77	93	109	125	141 5-8	157	173 11-2-8	189	205 11-5	221	237 11-5	253	D	LL0L
14	30	46	62	78	94	110 0-5-8	126	142	158	174 0-7-8	190	206 11-6	222	238 11-6	254	E	LLLO
15	31	47	63	79	95	111	127	143	159	175 keine Lochung	191 12-0	207 11-7	223	239 11-7	255	F	LLLL

wird durch Ersatzzeichen dargestellt

wird 12-11-1-2-3-4-5-6-7-8-9 zugeordnet

GR 35



Informationsverarbeitung
Lochkartencode

KC2

2N 0813

.112

1. Zweck und Anwendung

Der Lochkartencode KC2 entspricht der Symbolausführung EL (USA-Version) des IBM-Kartenlochers 29.

2. Code-Tabelle

Die Lochkartenzeilen werden von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet. Jedes darzustellende Zeichen wird in einer Spalte senkrecht zur Zeile gelocht.

In der Tabelle ist angegeben, welche Stellen einer Spalte gelocht sein müssen, um das Zeichen darzustellen. Die am Kopf der Tabelle und links der Tabelle angegebenen Stellen ergeben das Lochmuster für das Zeichen. Dabei bedeutet das Zeichen "-" keine Lochung.

Code - Tabelle

Lochung	12	11	0	-
-	&	-	0	SP
1	A	J	/	1
2	B	K	S	2
3	C	L	T	3
4	D	M	U	4
5	E	N	V	5
6	F	O	W	6
7	G	P	X	7
8	H	Q	Y	8
9	I	R	Z	9
2-8	¢	!	Leerstelle	:
3-8	.	\$,	#
4-8	<	*	%	@
5-8	()	-	'
6-8	+	;	>	=
7-8		¬	?	"

GR 179

Erläuterungen

Die Lochungen entsprechen denen des Kartenlochers 29 der IBM.

Gegenüber dem Zeichensatz ZS 64 (Werknorm 2N 0810.111) wurden beim Lochkartencode KC2 folgende Zeichen geändert:

ZS 64	KC2
À	#
Ö	@
Ü	\$
["
]	¢
⌘	Leerstelle

N3/NM

N31/V23

Ersatz für:

Ersetzt durch:

V

Frühere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:

Rm



Informationsverarbeitung
Umcodierleistung Lochkarteneingabe TR 440
 KC2 → ZC1

2N 0815

.112

1. Zweck und Anwendung

Der Karteneingabecode KEC2 stellt eine Zuordnung von Lochkombinationen zu Bitmustern von 8 Bits dar.

Durch den Karteneingabecode KEC2 wird, ausgehend vom Kartencode KC2 (Werknorm 2N 0813.112) eine Zuordnung gemäß Zentralcode ZC1 (Werknorm 2N 0812.511) erreicht.

Der vorliegende Karteneingabecode hat folgende Kennzeichnungen:

symbolischer Name bei der Steuerung der Eingabeprozedur: KC2

Kennzeichen für Codeeinstellung (siehe Abschnitt 4.1 und 5.1): 2

bezieht sich auf Lochkartencode (Werknorm 2N 0813.112): KC2

2. Bezeichnung

Für die Darstellung des Lochmusters einer Spalte werden die Zeilen der Lochkarte von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet.

In der Code-Tabelle ist jeweils oben im Feld das Zeichen angegeben, das der Lochkombination gemäß dem Kartencode KC2 (Werknorm 2N 0813.112) zugeordnet ist. Im unteren Teil des Feldes ist der dezimale Wert der Oktade angegeben, der diese Lochkombination zugeordnet wird.

3. Zuordnung

Die Zuordnung ist der Code-Tabelle zu entnehmen. Den Lochkombinationen, denen in der Code-Tabelle kein Dezimalwert zugeordnet ist, wird die Oktade mit dem Dezimalwert 32 zugeordnet. Außer der in der Code-Tabelle angegebenen, werden folgende Zuordnungen gemacht:

	Oktade	
	Symbol	dezimal
Kartende	NL	21
Abschnittsende	EM	33
Keine Zuordnung in der Code-Tabelle angegeben	SUB	32

4. Fluchtsymbol4.1. Codeunabhängiges Fluchtsymbol

Das codeunabhängige Fluchtsymbol wird durch die Lochung

12-11-5-8 in der Spalte 1

dargestellt. Es wird stets unmittelbar von dem Kennzeichen für die Codeeinstellung (siehe Abschnitt 1) gefolgt. Es wird nur zur Einleitung der Kommandos für die Eingabeprozedur verwendet (siehe Abschnitt 5.1).

4.2. Codeabhängiges Fluchtsymbol

Das codeabhängige Fluchtsymbol ist für den vorliegenden Code die Lochung

0-2-8 in der Spalte 1

Ihr ist im Kartencode KC2 das Zeichen "Leerstelle" zugeordnet. In dem in Abschnitt 5.2 angegebenen Fall darf das codeabhängige Fluchtsymbol in jeder Spalte stehen.

Des weiteren kann durch die Eingabeprozedur festgelegt werden, daß das codeabhängige Fluchtsymbol in jeder Spalte zugelassen ist.

5. Spezielle Folgen

In den nachstehenden Fällen wird eine andere Zuordnung von Lochmustern zu Oktaden vorgenommen, als in Abschnitt 3 angegeben.

5.1. Anweisungen an die Eingabeprozedur

Folgt unmittelbar auf ein codeunabhängiges Fluchtsymbol mit nachfolgendem Kennzeichen für die Codeeinstellung oder - soweit dies in der Eingabeprozedur vorgesehen ist - auf ein codeabhängiges Fluchtsymbol das Zeichen

X (Lochung 0-7)

so ist dies eine Anweisung an die Eingabeprozedur. Ein codeabhängiges Fluchtsymbol mit nachfolgendem Punkt beendet die Folge; in diesem Fall ist das codeabhängige Fluchtsymbol in jeder Spalte zugelassen.

Die Zuordnung der Lochungen der vorstehenden Anweisungen zu Oktaden wird durch die Eingabeprozedur festgelegt.

N3/N**N 31/V23**

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2

V

Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

ungen:

Rm

5.2. Dezimaler Oktadenwert

Falls die Darstellung einer beliebigen Oktade durch ihren dezimalen Wert in der Eingabeprozedur vorgesehen ist, gilt nachstehende Festlegung.

Folgen einem codeabhängigen Fluchtsymbol drei Ziffern (zugehörige Lochungen siehe Code-Tabelle), so stellt diese dreistellige Dezimalzahl einen Oktadenwert dar. Der Folge (Fluchtsymbol, drei Ziffern) wird der angegebene Oktadenwert zugeordnet.

◇ 0 9 6 } ergibt } 96 }

◇ entspricht der Lochung 0-2-8 (Leerstelle)

Es sind nur die Werte von 064 bis 255 zugelassen.
Alle 4 Zeichen müssen in derselben Lochkarte sein.

Erläuterungen

Durch diese Norm wird die Steuerung der Eingabeprozedur (siehe Abschnitt 5.1) nicht festgelegt.

Auch die Möglichkeit, daß durch das Kommando die Eingabeprozedur so umgesteuert werden kann, daß eine andere als die hier festgelegte Zuordnung erreicht werden kann und auch, daß auf einen anderen Code umgeschaltet werden kann, wurde hier nicht angegeben.

Code-Tabelle

Lochung	12	11	0	-	12-0	12-11	11-0	12-11-0
-	& 119	- 145	0 176	SP 175	PZ 191		MZ 168	
1	A 192	J 201	/ 147	1 177				
2	B 193	K 202	S 210	2 178				
3	C 194	L 203	T 211	3 179				
4	D 195	M 204	U 212	4 180				
5	E 196	N 205	V 213	5 181				
6	F 197	O 206	W 214	6 182				
7	G 198	P 207	X 215	7 183				
8	H 199	Q 208	Y 216	8 184				
9	I 200	R 209	Z 217	9 185				
2-8	∅ 116	! 173	Leerstelle 53	: 171				
3-8	. 169	¢ 115	, 170	# 114				
4-8	< 155	* 120	% 112	@ 118				
5-8	(160) 161	170	' 97				
6-8	+ 144	; 172	> 156	= 151				
7-8	 139	¬ 130	? 174	" 96				

GR 176



Informationsverarbeitung
Umcodierleistung Lochkartenausgabe TR 440
 ZC1 → KC2

2N 0815

.212

Einsprüche bis 31. August 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3: Codes, Schriftführer: Böckmann N31/V23, Tel. 2474

1. Zweck und Anwendung

Diese Werknorm legt fest, welche Umcodierleistung bei der Lochkartenausgabe des TR 440 im Code KC2 erbracht wird. Den Oktaden des Zentralcodes ZC1 (Werknorm 2N 0812.511) werden Lochmuster des Lochkartencodes KC2 (Werknorm 2N 0813.112) zugeordnet.

2. Bezeichnung

In der Code-Tabelle wird vom Bitmuster des Zentralcodes ZC1 ausgegangen. In der Kopfleiste und der rechten Seitenleiste sind Dualwert und Sedezimalwerte des ZC1 und in der linken oberen Ecke eines jeden Feldes der Dezimalwert der ZC1-Oktade angegeben. Die weiter im Feld vorhandene Angabe bezeichnet das Lochmuster, das der ZC1-Oktade zugeordnet wird. Die Lochkartenzeilen sind von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet.

Die Übersichtstabelle entspricht dem Zentralcode ZC1 und gibt eine Übersicht über die Zeichen, die auf dem Drucker gemäß dieser Werknorm dargestellt werden können. Sie ist nicht Bestandteil dieser Werknorm.

3. Informationsdarstellung

Die auszugebende Information hat die Form einer Datei vom Typ sequentiell. Jeder Satz der Datei bildet eine Karte. Die Information der Datei kann im A-Format oder im O-Format vorliegen.

3.1. A-Format

Das erste Zeichen jedes Satzes wird ignoriert.

3.2. O-Format

Das erste Zeichen jedes Satzes bildet das erste Zeichen der Lochkarte.

3.3. Länge eines Satzes

Enthält ein Satz weniger als 80 Zeichen, so bleibt der Rest der Karte leer (auffüllen mit SP).

Enthält ein Satz mehr als 80 Zeichen, so werden die überzähligen Zeichen so behandelt, als wären sie nicht vorhanden.

4. Zuordnung

Die Zeichen des Zentralcodes werden gemäß der Code-Tabelle dem Lochmuster zugeordnet. Für Zeichen, deren Felder punktiert sind, wird ein Ersatzzeichen ausgegeben. Für das Zeichen Space (dez. 175) wird keine Lochung ausgegeben. Für alle Zeichen, denen in der Code-Tabelle keine Lochung zugeordnet ist, wird das Lochmuster 12-11-0-1-2-3-4-5-6-7-8-9 (vollständig durchlochte Spalte) ausgegeben.

ZC1			KC2	
Zeichen	dez.		Zeichen	Lochg.
FL Fluchtsymbol	53	χ	codeunabhg. Fluchtsymbol	5-8-11-12
NUL Nil	0		keine Zuordnung	
NU1	1		keine Zuordnung	
•	•		keine Zuordnung	
•	•		keine Zuordnung	
•	•		keine Zuordnung	
NU5	5		keine Zuordnung	
₁₀ Basiszehn	141	'	Apostroph	5-8
alle nicht angegebenen ZC1-Zeichen			Durchlochung	
Kleinbuchstaben			Großbuchstaben*)	

5. Sonstiges

Bei der Ausgabe werden in der nachstehenden Reihenfolge ausgegeben:

- 1 Leerkarte
- 1 Klarschriftkarte
- Informationskarten
- 1 Klarschriftendekarte

Durch Aussteuern einer Leerkarte werden Abschnitte bzw. Dateien optisch getrennt.

5.1. Leerkarte

Eine Leerkarte enthält keine Lochung. (Alle Spalten enthalten SP.)

5.2. Klarschriftkarte

Die Klarschriftkarte enthält ein Lochmuster, das Klarschrift darstellt und vom Bedienungspersonal für die Identifizierung gelesen werden kann. Die Klarschrift beginnt ab Spalte 5. Die Klarschriftendekarte enthält darüber hinaus eine Durchlochung in den Spalten 79 und 80.

*) Es steht noch nicht fest, ob den Kleinbuchstaben Großbuchstaben oder Durchlochungen zugeordnet werden.

N3/N

N 31/V23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2 und 3

V

are Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig missbräuchlich benutzt werden

erungen:

Ru

Übersichts - Tabelle

ZC1 → KC2

2⁷ 2⁰
 1. Sede-
 zimale 2. Sede-
 zimale

0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	LOLL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0	NUL	NL6	SUB			"	%	^	+	(0	A	Q	a	q	0	0000
1	NU1	NL5	EM			,	§	v	-)	1	B	R	b	r	1	000L
2	NU2	NL4				/	#	┘		[2	C	S	c	s	2	00LO
3	NU3	NL3				\	\$	/]	3	D	T	d	t	3	00LL
4	NU4	NL2				°	¢	{		}	4	E	U	e	u	4	0LOO
5	NU5	NL	TE	FL		^		}		}	5	F	V	f	v	5	0LOL
6	CR					°	@	†		{	6	G	W	g	w	6	0LLO
7	NF					-	&	=		}	7	H	X	h	x	7	0LLL
8	VT						*	≠	MZ		8	I	Y	i	y	8	L000
9	VT3								.		9	J	Z	j	z	9	L00L
10	VT4								,			K	Ä	k	ä	A	LOLO
11	VT5					/			<	:		L	Ö	l	ö	B	LOLL
12	VT6					∩	◻	>	;			M	Ü	m	ü	C	LL00
13	VT7)		∩	!			N		n		D	LL0L
14	VT8					-		∩	?			O		o		E	LLLO
15	NL7					-	π				SP	PZ	P	p		F	LLLL



wird durch Ersatzzeichen dargestellt

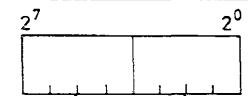


nicht erlaubte Zeichen

GR 2a

Code - Tabelle

ZC1 → KC2



0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	L0LL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	0	0000
1	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241	1	000L
2	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242	2	00LO
3	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243	3	00LL
4	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244	4	0LOO
5	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245	5	0LOL
6	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246	6	0LLO
7	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247	7	0LLL
8	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248	8	L000
9	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249	9	L00L
10	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250	A	L0LO
11	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251	B	L0LL
12	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252	C	LL00
13	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253	D	LL0L
14	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254	E	LLLO
15	31	47	63	79	95	111	127	143	159	175 keine Lochung	191	207	223	239	255	F	LLLL



wird durch Ersatzzeichen dargestellt



wird 12-11-0-1-2-3-4-5-6-7-8-9 zugeordnet

IGR 35



Informationsverarbeitung
Lochkartencode
KC3

2N 0813

.113

1. Zweck und Anwendung

Der Lochkartencode KC3 wurde von der DFG (Deutsche Forschungsgemeinschaft) empfohlen und ist speziell für die Benutzer von ALGOL gedacht.

Der Code enthält als Untermenge die Lochungen nach DIN 66006 ("Darstellung von ALGOL-Symbolen auf 80spaltigen Lochkarten") zuzüglich 16 weiteren algol-spezifischen Zeichen, die in der Tabelle in den Zeilen 2-8, 5-8, 6-8 und 7-8 stehen und durch Schraffur gekennzeichnet sind. (Bezüglich des Dollarzeichens \$ siehe Karteneingabecode KEC3, Werknorm 2N 0815.113).

Im Lochkartencode KC3 sind alle Zeichen des Zeichensatzes ZS 48 (Werknorm 2N 0810.111) enthalten.

Code - Tabelle

Lochung	12	11	0	-
-	+	-	0	SP
1	A	J	/	1
2	B	K	S	2
3	C	L	T	3
4	D	M	U	4
5	E	N	V	5
6	F	O	W	6
7	G	P	X	7
8	H	Q	Y	8
9	I	R	Z	9
2-8	≤	>	≠	≥
3-8	.	\$,	=
4-8)	*	('
5-8	[]	10	:
6-8	<	,	ρ	>
7-8	↑	→	∇	∧

2. Code-Tabelle

Die Lochkartenzeilen werden von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet. Jedes darzustellende Zeichen wird in einer Spalte senkrecht zur Zeile gelocht.

In der Tabelle ist angegeben, welche Stellen einer Spalte gelocht sein müssen, um das Zeichen darzustellen. Die am Kopf der Tabelle und links der Tabelle angegebenen Stellen ergeben das Lochmuster für das Zeichen. Dabei bedeutet das Zeichen "-" keine Lochung.

Erläuterungen

Der Lochkartencode KC3 enthält als Untermenge den Zeichenvorrat der Symbolausführung H der IBM für Kartenlocher 29 bzw. 26 (in der Tabelle die Zeichen in den Zeilen -, 1 bis 9, 3-8 und 4-8).

N3/NM

N31/V23

Ersatz für:

Ersetzt durch:

V

frühere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:

Rm

GR 180



Informationsverarbeitung
Umcodierleistung Lochkarteneingabe TR 440
 KC3 → ZC1

2N 0815

.113

1. Zweck und Anwendung

Der Karteneingabecode KEC3 stellt eine Zuordnung von Lochkombinationen zu Bitmustern von 8 Bits dar.

Durch den Karteneingabecode KEC3 wird, ausgehend vom Kartencode KC3 (Werknorm 2N 0813.113) eine Zuordnung gemäß Zentralcode ZC1 (Werknorm 2 N 0812.511) erreicht.

Der vorliegende Karteneingabecode hat folgende Kennzeichnungen:

- symbolischer Name bei der Steuerung der Eingabeprozedur: KC3
- Kennzeichen für Codeeinstellung (siehe Abschnitt 4.1 und 5.1): 3
- bezieht sich auf Lochkartencode (Werknorm 2N 0813.113): KC3

2. Bezeichnung

Für die Darstellung des Lochmusters einer Spalte werden die Zeilen der Lochkarte von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet.

In der Code-Tabelle ist jeweils oben im Feld das Zeichen angegeben, das der Lochkombination gemäß dem Kartencode KC3 (Werknorm 2N 0813.113) zugeordnet ist. Im unteren Teil des Feldes ist der dezimale Wert der Oktade angegeben, der diese Lochkombination zugeordnet wird.

3. Zuordnung

Die Zuordnung ist der Code-Tabelle zu entnehmen. Den Lochkombinationen, denen in der Code-Tabelle kein Dezimalwert zugeordnet ist, wird die Oktade mit dem Dezimalwert 32 zugeordnet. Außer der in der Code-Tabelle angegebenen, werden folgende Zuordnungen gemacht:

	Oktade	
	Symbol	dezimal
Kartencode	NL	21
Abschnittsende	EM	33
Keine Zuordnung in der Code-Tabelle angegeben	SUB	32

4. Fluchtsymbol

4.1. Codeunabhängiges Fluchtsymbol

Das codeunabhängige Fluchtsymbol wird durch die Lochung

12-11-5-8 in der Spalte 1

dargestellt. Es wird stets unmittelbar von dem Kennzeichen für die Codeeinstellung (siehe Abschnitt 1) gefolgt. Es wird nur zur Einleitung der Kommandos für die Eingabeprozedur verwendet (siehe Abschnitt 5.1).

4.2. Codeabhängiges Fluchtsymbol

Das codeabhängige Fluchtsymbol ist für den vorliegenden Code die Lochung

11-3-8 in der Spalte 1

Ihr ist im Kartencode KC3 das Zeichen \$ zugeordnet. In dem in Abschnitt 5.2 angegebenen Fall darf das codeabhängige Fluchtsymbol in jeder Spalte stehen.

Des weiteren kann durch die Eingabeprozedur festgelegt werden, daß das codeabhängige Fluchtsymbol in jeder Spalte zugelassen ist.

5. Spezielle Folgen

In den nachstehenden Fällen wird eine andere Zuordnung von Lochmustern zu Oktaden vorgenommen, als in Abschnitt 3 angegeben.

5.1. Anweisungen an die Eingabeprozedur

Folgt unmittelbar auf ein codeunabhängiges Fluchtsymbol mit nachfolgendem Kennzeichen für die Codeeinstellung, oder - soweit dies in der Eingabeprozedur vorgesehen ist - auf ein codeabhängiges Fluchtsymbol das Zeichen

X (Lochung 0-7)

so ist dies eine Anweisung an die Eingabeprozedur. Ein codeabhängiges Fluchtsymbol mit nachfolgendem Punkt beendet die Folge; in diesem Fall ist das codeabhängige Fluchtsymbol in jeder Spalte zugelassen.

Die Zuordnung der Lochungen der vorstehenden Anweisungen zu Oktaden wird durch die Eingabeprozedur festgelegt.

Diese Unterlage darf weder kopiert, noch
 Dritten Personen mitgeteilt, noch ander-
 weitig mißbräuchlich benutzt werden.

Änderungen:

Rm

5.2. Dezimaler Oktadenwert

Falls die Darstellung einer beliebigen Oktade durch ihren dezimalen Wert in der Eingabeprozedur vorgesehen ist, gilt nachstehende Festlegung.

Folgen einem codeabhängigen Fluchtsymbol drei Ziffern (zugehörige Lochungen siehe Code-Tabelle), so stellt diese dreistellige Dezimalzahl einen Oktadenwert dar. Der Folge (Fluchtsymbol, drei Ziffern) wird der angegebene Oktadenwert zugeordnet.

\$	0	9	6
----	---	---	---

 ergibt

96

Es sind nur die Werte von 064 bis 255 zugelassen.
Alle 4 Zeichen müssen in derselben Lochkarte sein.

Erläuterungen

Durch diese Norm wird die Steuerung der Eingabeprozedur (siehe Abschnitt 5.1) nicht festgelegt.

Auch die Möglichkeit, daß durch das Kommando die Eingabeprozedur so umgesteuert werden kann, daß eine andere als die hier festgelegte Zuordnung erreicht werden kann und auch, daß auf einen anderen Code umgeschaltet werden kann, wurde hier nicht angegeben.

Code - Tabelle

Lochung	12	11	0	-	12-0	12-11	11-0	12-11-0
-	+ 144	- 145	0 176	SP 175	PZ 191		MZ 168	
1	A 192	J 201	/ 147	1 177				
2	B 193	K 202	S 210	2 178				
3	C 194	L 202	T 211	3 179				
4	D 195	M 204	U 212	4 180				
5	E 196	N 205	V 213	5 181				
6	F 197	O 206	W 214	6 182				
7	G 198	P 207	X 215	7 183				
8	H 199	Q 208	Y 216	8 184				
9	I 200	R 209	Z 217	9 185				
2-8	≤ 157	\ 109	≠ 152	≥ 158				
3-8	· 169	§ 53	, 170	= 151				
4-8) 161	* 120) 160	' 97				
5-8	[162] ¹⁰ 163	¹⁰ 141	: 171				
6-8	< 155	; 172	(108	> 156				
7-8	↑ 134	¬ 130	∨ 129	∧ 128				

GR 177



Informationsverarbeitung
Umcodierleistung Lochkartenausgabe TR 440
 ZC1 → KC3

2N 0815

.213

Einsprüche bis 31. August 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3: Codes, Schriftführer: Böckmann N31/V23, Tel. 2474

1. Zweck und Anwendung

Diese Werknorm legt fest, welche Umcodierleistung bei der Lochkartenausgabe des TR 440 im Code KC3 erbracht wird. Den Oktaden des Zentralcodes ZC1 (Werknorm 2N 0812.511) werden Lochmuster des Lochkartencodes KC3 (Werknorm 2N 0813.113) zugeordnet.

2. Bezeichnung

In der Code-Tabelle wird vom Bitmuster des Zentralcodes ZC1 ausgegangen. In der Kopfleiste und der rechten Seitenleiste sind Dualwert und Sedezimalwerte des ZC1 und in der linken oberen Ecke eines jeden Feldes der Dezimalwert der ZC1-Oktade angegeben. Die weiter im Feld vorhandene Angabe bezeichnet das Lochmuster, das der ZC1-Oktade zugeordnet wird. Die Lochkartenzeilen sind von oben nach unten mit 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 bezeichnet.

Die Übersichtstabelle entspricht dem Zentralcode ZC1 und gibt eine Übersicht über die Zeichen, die auf dem Drucker gemäß dieser Werknorm dargestellt werden können. Sie ist nicht Bestandteil dieser Werknorm.

3. Informationsdarstellung

Die auszugebende Information hat die Form einer Datei vom Typ sequentiell. Jeder Satz der Datei bildet eine Karte. Die Information der Datei kann im A-Format oder im O-Format vorliegen.

3.1. A-Format

Das erste Zeichen jedes Satzes wird ignoriert.

3.2. O-Format

Das erste Zeichen jedes Satzes bildet das erste Zeichen der Lochkarte.

3.3. Länge eines Satzes

Enthält ein Satz weniger als 80 Zeichen, so bleibt der Rest der Karte leer (auffüllen mit SP).

Enthält ein Satz mehr als 80 Zeichen, so werden die überzähligen Zeichen so behandelt, als wären sie nicht vorhanden.

4. Zuordnung

Die Zeichen des Zentralcodes werden gemäß der Code-Tabelle dem Lochmuster zugeordnet. Für Zeichen, deren Felder punktiert sind, wird ein Ersatzzeichen ausgegeben. Für das Zeichen Space (dez. 175) wird keine Lochung ausgegeben. Für alle Zeichen, denen in der Code-Tabelle keine Lochung zugeordnet ist, wird das Lochmuster 12-11-0-1-2-3-4-5-6-7-8-9 (vollständig durchlochte Spalte) ausgegeben.

ZC1			KC3	
Zeichen	dez.		Zeichen	Lochg.
FL Fluchtsymbol	53		λ codeunabhg. Fluchtsymbol	12-11-5-8
NUL Nil	0		keine Zuordnung	
NU1	1		keine Zuordnung	
.	.		keine Zuordnung	
.	.		keine Zuordnung	
NU5	5		keine Zuordnung	
alle nicht angegebenen ZC1-Zeichen			Durchlochung	
Kleinbuchstaben			Großbuchstaben*)	

5. Sonstiges

Bei der Ausgabe werden in der nachstehenden Reihenfolge ausgegeben:

- 1 Leerkarte
- 1 Klarschriftkarte
- Informationskarten
- 1 Klarschriftendekarte

Durch Aussteuern einer Leerkarte werden Abschnitte bzw. Dateien optisch getrennt.

5.1. Leerkarte

Eine Leerkarte enthält keine Lochung. (Alle Spalten enthalten SP).

5.2. Klarschriftkarte

Die Klarschriftkarte enthält ein Lochmuster, das Klarschrift darstellt und vom Bedienungspersonal für die Identifizierung gelesen werden kann. Die Klarschrift beginnt ab Spalte 5. Die Klarschriftendekarte enthält darüber hinaus eine Durchlochung in den Spalten 79 und 80.

*) Es steht noch nicht fest, ob den Kleinbuchstaben Großbuchstaben oder Durchlochungen zugeordnet werden.



re Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Ergebnis:

Rm


Übersichts - Tabelle

ZC1 → KC3

2⁷ 2⁰
1. Sede- 2. Sede-
zimale zimale

0000	000L	00LO	00LL	0LO0	0LOL	0LLO	0LLL	L000	L00L	LOLO	LOLL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0 NUL	16 NL6	32 SUB	48 .	64 .	80 .	96 "	112 %	128 ^	144 +	160 (176 0	192 A	208 Q	224 a	240 q	0	0000
1 NU1	17 NL5	33 EM	49 .	65 .	81 .	97 '	113 \$	129 v	145 -	161)	177 1	193 B	209 R	225 b	241 r	1	000L
2 NU2	18 NL4	34 .	50 .	66 .	82 .	98 /	114 #	130 ¬	146 .	162 [178 2	194 C	210 S	226 c	242 s	2	00LO
3 NU3	19 NL3	35 .	51 .	67 .	83 .	99 \	115 \$	131 .	147 /	163]	179 3	195 D	211 T	227 d	243 t	3	00LL
4 NU4	20 NL2	36 .	52 .	68 .	84 .	100 .	116 ¢	132 .	148 .	164 {	180 4	196 E	212 U	228 e	244 u	4	0L00
5 NU5	21 NL	37 TE	53 FL	69 .	85 .	101 ^	117 .	133 .	149 .	165 }	181 5	197 F	213 V	229 f	245 v	5	0LOL
6 CR	22 .	38 .	54 .	70 .	86 .	102 o	118 @	134 ↑	150 .	166 <	182 6	198 G	214 W	230 g	246 w	6	0LLO
7 NF	23 .	39 .	55 .	71 .	87 .	103 -	119 &	135 .	151 =	167 >	183 7	199 H	215 X	231 h	247 x	7	0LLL
8 VT	24 .	40 .	56 .	72 .	88 .	104 .	120 *	136 .	152 ≠	168 MZ	184 8	200 I	216 Y	232 i	248 y	8	L000
9 VT3	25 .	41 .	57 .	73 .	89 .	105 .	121 .	137 .	153 .	169 .	185 9	201 J	217 Z	233 j	249 z	9	L00L
10 VT4	26 .	42 .	58 .	74 .	90 .	106 .	122 .	138 .	154 .	170 ,	186 .	202 K	218 A	234 k	250 a	A	LOLO
11 VT5	27 .	43 .	59 .	75 .	91 .	107 \	123 .	139 .	155 <	171 :	187 .	203 L	219 O	235 l	251 o	B	LOLL
12 VT6	28 .	44 .	60 .	76 .	92 .	108 '	124 H	140 .	156 >	172 ;	188 .	204 M	220 U	236 m	252 u	C	LL00
13 VT7	29 .	45 .	61 .	77 .	93 .	109 '	125 .	141 10	157 ≡	173 !	189 .	205 N	221 .	237 n	253 ß	D	LL0L
14 VT8	30 .	46 .	62 .	78 .	94 .	110 .	126 .	142 .	158 ≡	174 ?	190 .	206 O	222 .	238 o	254 .	E	LLLO
15 NL7	31 .	47 .	63 .	79 .	95 .	111 -	127 π	143 .	159 .	175 SP	191 PZ	207 P	223 .	239 p	255 .	F	LLLL

 wird durch Ersatzzeichen dargestellt

 nicht erlaubte Zeichen

GR 2a

Code - Tabelle

ZC1 → KC3



0000	000L	00LO	00LL	OL00	OL0L	OLLO	OLLL	L000	L00L	L0LO	L0LL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0	16	32	48	64	80	96	112	128 7-8	144 12	160 0-4-8	176 0	192 12-1	208 11-8	224 12-1	240 11-8	0	0000
1	17	33	49	65	81	97 4-8	113	129 0-7-8	145 11	161 12-4-8	177 1	193 12-2	209 11-9	225 12-2	241 11-9	1	000L
2	18	34	50	66	82	98	114	130 11-7-8	146	162 12-5-8	178 2	194 12-3	210 0-2	226 12-3	242 0-2	2	00LO
3	19	35	51	67	83	99	115 11-3-8	131	147 0-1	163 11-5-8	179 3	195 12-4	211 0-3	227 12-4	243 0-3	3	00LL
4	20	36	52	68	84	100	116	132	148	164	180 4	196 12-5	212 0-4	228 12-5	244 0-4	4	OL00
5	21	37	53 12-11-5-8	69	85	101	117	133	149	165	181 5	197 12-6	213 0-5	229 12-6	245 0-5	5	OL0L
6	22	38	54	70	86	102	118	134 12-7-8	150	166	182 6	198 12-7	214 0-6	230 12-7	246 0-6	6	OLLO
7	23	39	55	71	87	103	119	135	151 3-8	167	183 7	199 12-8	215 0-7	231 12-8	247 0-7	7	OLLL
8	24	40	56	72	88	104	120 11-4-8	136	152 0-2-8	168 11-0	184 8	200 12-9	216 0-8	232 12-9	248 0-8	8	L000
9	25	41	57	73	89	105	121	137	153	169 12-3-8	185 9	201 11-1	217 0-9	233 11-1	249 0-9	9	L00L
10	26	42	58	74	90	106	122	138	154	170 0-3-8	186	202 11-2	218	234 11-2	250	A	LOLO
11	27	43	59	75	91	107	123	139	155 12-6-8	171 5-8	187	203 11-3	219	235 11-3	251	B	LOLL
12	28	44	60	76	92	108 0-6-8	124	140	156 6-8	172 11-6-8	188	204 11-4	220	236 11-4	252	C	LL00
13	29	45	61	77	93	109 11-2-8	125	141 0-5-8	157 12-2-8	173	189	205 11-5	221	237 11-5	253	D	LL0L
14	30	46	62	78	94	110	126	142	158 2-8	174	190	206 11-6	222	238 11-6	254	E	LLLO
15	31	47	63	79	95	111	127	143	159	175 keine Lochung	191 12-0	207 11-7	223	239 11-7	255	F	LLLL

wird durch Ersatzzeichen dargestellt

wird 12-11-0-1-2-3-4-5-6-7-8-9 zugeordnet

GR 35



Informationsverarbeitung
Lochstreifencode
SC1

2N 0813

.211

1. Zweck und Anwendung

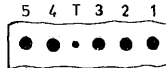
Der Lochstreifencode SC1 enthält den Zeichensatz ZS54 (siehe Werknorm 2N 0810.111). Er ist für Fernschreiber und Streifenleser vorgesehen.

2. Darstellung auf Lochstreifen

Für die Darstellung des Codes werden Lochstreifen nach DIN 66 016 verwendet. Der Code wird in den Spuren 1 bis 5 dargestellt. Für die binäre Darstellung wird der Spur 1 das Bit mit der Wertigkeit 2⁰ zugeordnet.

3. Code-Tabelle

Bu		Zi		
0	1	0	1	
&	E	⊠	3	0
T	Z	5	+	1
CR	D	CR	⊞	2
O	B	9	*	3
SP	S	SP	'	4
H	Y	!	6	5
N	F	,	[6
M	X	.	/	7
LF	A	LF	-	8
L	W)	2	9
R	J	4	;	A
G	ZI]	ZI	B
I	U	8	7	C
P	Q	0	1	D
C	K	:	(E
V	BU	=	BU	F



4. Steuerzeichen

CR Wagenrücklauf (Carriage Return)

Das Zeichen dient zur Rückbewegung der Schreibeinrichtung auf die erste Schreibposition der gleichen Zeile.

LF Zeilenvorschub (Line Feed)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung auf die gleiche Schreibposition der nächsten Zeile.

BU Buchstabenumschaltung

Alle folgenden Zeichen sind nach der Buchstabenseite (Bu) der Code-Tabelle zu interpretieren.

ZI Ziffernumschaltung

Alle folgenden Zeichen sind nach der Zifferseite (Zi) der Code-Tabelle zu interpretieren.

SP Zwischenraum (Space)

Das Zeichen dient zur Vorwärtsbewegung der Schreibeinrichtung um eine Schreibposition.

⊞ Wer da?

Das Zeichen löst den fremden Namengeber aus.

Erläuterungen

Ausgangspunkt für diesen Code ist das internationale Telegraphen-Alphabet Nr. 2 der CCITT und die Darstellung von ALGOL-Symbolen auf Lochstreifen nach DIN 66 006. In den folgenden Punkten wurde davon abgewichen:

SC1	DIN 66 006	CCITT 2
*	x	?
;	;	⊞
[[
]]	
!	!	
&		
⊞		
⊞		⊞

N3/N

N31/V23

Ersatz für:

Ersetzt durch:

V

Frei-Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Anmerkungen:

Rm



Informationsverarbeitung
Umcodierleistung Lochstreifeneing. TR440
 SC1 → ZC1

2N 0815

.411

Einsprüche bis 31. März 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3: C o d e s, Schriftführer: Böckmann N31/V23, Tel 474

1. Zweck und Anwendung

Code-Tabelle

Diese Werknorm legt fest, welche Umcodierleistung bei

- Lochstreifeneingabe über Streifenleser,
- Eingabe über die Fernschreibtastatur,
- Eingabe über den Lochstreifenleser des Fernschreibers

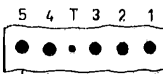
erbracht werden.

Den Zeichen des Lochstreifencodes SC 1 (Werknorm 2N 0813.211) werden Oktaden des Zentralcodes ZC 1 (Werknorm 2N 0812.511) zugeordnet.

Für die vorliegende Norm gelten folgende Kennzeichnungen:

- Symbolischer Name bei der Steuerung der Eingabeprozedur: SC1
- Kennzeichen der Codeeinstellung (s. Abschn. 4.1 und 5.1): BU '01'
- Bezieht sich auf Lochstreifencode (Werknorm 2N 0813.211): SC1

Bu		Zi		
0	1	0	1	
& 119	E 196	≡ 53	3 179	0
T 211	Z 217	5 181	+ 144	1
CR	D 195	CR	⊕	2
O 206	B 193	9 185	* 120	3
SP 175	S 210	SP 175	' 97	4
H 199	Y 216	I 173	6 182	5
N 205	F 197	7 170	Γ 162	6
M 204	X 215	. 169	/ 147	7
LF 21	A 192	LF 21	- 145	8
L 203	W 214) 161	2 178	9
R 209	J 201	4 180	; 172	A
G 198	ZI	J 163	ZI	B
I 200	U 212	8 184	7 183	C
P 207	Q 208	0 176	1 177	D
C 194	K 202	: 171	(160	E
V 213	BU	= 151	BU	F



keine Zuordnung

2. Darstellung

In den Feldern der Code-Tabelle ist jeweils oben das Zeichen angegeben, das der Lochkombination des Lochstreifencodes SC1 zugeordnet ist. Im unteren Teil des Feldes ist der dezimale Wert der Oktade angegeben, der diese Lochkombination zugeordnet wird.

Die Zeichen BU und ZI dienen der Umschaltung auf die linke bzw. rechte Hälfte der Code-Tabelle und werden nicht einer Oktade zugeordnet.

BU und ZI sind Steuerzeichen. BU und Zi bezeichnen die Hälften der Tabelle.

Soweit im Text zwei Zeichen von Apostrophen eingeschlossen sind, handelt es sich um die sedezimale Darstellung der Lochkombination.

3. Zuordnung

Die Zuordnung ist der Code-Tabelle zu entnehmen. Eine andere Zuordnung ergibt sich in den im Abschnitt 5 aufgeführten Fällen. Das Zeichen CR wird ignoriert.

4. Fluchtsymbol

4.1. Codeunabhängiges Fluchtsymbol (FU)

Das codeunabhängige Fluchtsymbol besteht aus

- Leerband (mindestens 1 Zeichen)
- 5 Zeichen BU ('1F')

Es folgt stets unmittelbar das Kennzeichen für die Codeeinstellung (s. Abschn. 1). Das codeunabhängige Fluchtsymbol wird nur zur Einleitung der Kommandos für die Eingabeprozedur verwendet (s. Abschnitt 5.1).

Frühere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:

4.2. Codeabhängiges Fluchtsymbol (Fa)

Das codeabhängige Fluchtsymbol ist für den vorliegenden Code das Zeichen

▣ (ZI '00')

(Anmerkung: Die Darstellung auf dem Lochstreifen ist das Leerband. Wird beim Einlegen eines Lochstreifens das Leerband vor die Lesestation gelegt, so werden codeabhängige Fluchtsymbole eingelesen, falls ZI eingestellt ist.)

Mehrere aufeinanderfolgende Fluchtsymbole werden als ein Fluchtsymbol behandelt.

5. Spezielle Folgen

In den nachstehenden Fällen wird eine andere Zuordnung vorgenommen als in Abschnitt 3 angegeben.

5.1. Anweisungen an die Eingabeprozedur

Folgt unmittelbar auf ein codeunabhängiges Fluchtsymbol (Fu) mit nachfolgendem Kennzeichen für die Codeeinstellung das Zeichen

X (BU' 17')

oder auf ein codeabhängiges Fluchtsymbol (Fa) das Zeichen

X (BU' 17')
((ZI' 1E')

so ist dies eine Anweisung an die Eingabeprozedur. Ein codeabhängiges Fluchtsymbol (Fa) mit nachfolgendem Punkt beendet diese Anweisung.

Die Zuordnung der Lochungen der vorstehenden Anweisungen zu Oktaden wird durch die Eingabeprozedur festgelegt.

5.2. Dezimaler Oktadenwert

Falls die Darstellung einer beliebigen Oktade durch ihren dezimalen Wert in der Eingabeprozedur vorgesehen ist, gilt nachstehende Festlegung.

Folgt einem codeabhängigen Fluchtsymbol drei Ziffern (zugehörige Lochungen siehe Code-Tabelle), so stellt diese dreistellige Dezimalzahl einen Oktadenwert dar. Der Folge (Fluchtsymbol, drei Ziffern) wird der angegebene Oktadenwert zugeordnet.

{▣ 0 9 6} ergibt {96}

Es sind nur die Werte von 064 bis 255 zugelassen.

5.3. Leerband

Das Leerband enthält nur die Transportlöcher der Taktspur. Es hat im vorliegenden Fall bei ZI-Einstellung die Bedeutung von codeabhängigem Fluchtsymbol. Bei der Eingabe über die Fernschreibtastatur entfällt das Leerband.

5.4. Vorspann, Nachspann

Bei Lochstreifen muß vor und hinter der Information ein Vorspann bzw. Nachspann vorhanden sein. Er besteht jeweils aus

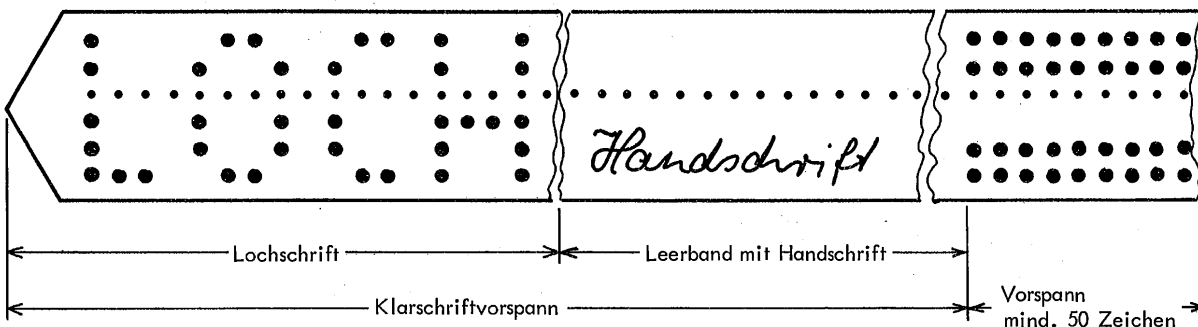
mindestens 50 Zeichen ZI .

Ein Lochstreifen muß so eingelegt werden, daß die Lesestation innerhalb des Vorspanns zu lesen beginnt.

Bei der Eingabe über den Lochstreifenleser des Fernschreibers muß die Eingabe abgebrochen werden, solange der Nachspann gelesen wird.

5.5. Klarschriftvorspann

Der Klarschriftvorspann liegt vor dem Vorspann und kann beliebige Information enthalten. Er darf jedoch nicht mit eingelesen werden.





Informationsverarbeitung
Umcodierleistung Lochstreifenausg. TR 440
 ZC1 → SC1

2N 0815

.511

Einsprüche bis 31. März 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3:Codes, Schriftführer: Böckmann N31/V23, Tel. 474

1. Zweck und Anwendung

Diese Werknorm legt fest, welche Umcodierleistung bei der Ausgabe über einen Fernschreiber und bei der Lochstreifenausgabe im Code SC1 des TR 440 erbracht wird. Den Oktaden des Zentralcodes ZC1 (Werknorm 2 N 0812.511) werden Lochmuster des Lochstreifencodes SC1 (Werknorm 2 N 0813.211) zugeordnet.

2. Bezeichnung

In der Code-Tabelle wird vom Bitmuster des Zentralcodes ZC1 ausgegangen. In der Kopfleiste und der rechten Seitenleiste sind Dualwert und Sedezimalwert des ZC1 und in der linken oberen Ecke eines jeden Feldes der Dezimalwert der ZC1-Oktade angegeben. Die Angabe im Feld verweist auf das zugehörige Zeichen im SC1.

Alle Angaben für das zugehörige Zeichen im Code SC1 bestehen aus einer zweistelligen Sedezimalzahl. Sie stellt das Lochmuster des SC1-Zeichens dar. Ein vorangestelltes "Bu" bedeutet, daß das Zeichen nach der Buchstabenseite der Code-Tabelle zu interpretieren ist; bei einem vorangestellten "Zi" ist das Zeichen nach der Ziffernseite zu interpretieren. Ist "Bu" oder "Zi" nicht angegeben, so hat das Zeichen sowohl auf der Buchstabenseite als auch auf der Ziffernseite dieselbe Bedeutung.

Die Übersichtstabelle entspricht dem Zentralcode ZC1 und gibt eine Übersicht über die Zeichen, die als Ausgabezeichen dargestellt werden können. Die Übersichtstabelle ist nicht Bestandteil dieser Werknorm.

3. Informationsdarstellung

Die auszugebende Information hat die Form einer Datei vom Typ sequentiell. Jede Druckzeile bildet einen Satz der Datei. Die Information einer Datei kann entweder im A-Format oder im O-Format vorliegen. Das Format bestimmt die Art der Vorschubsteuerung und ist bei der Erteilung des Ausgabeauftrags mit anzugeben.

3.1. A-Format

Das erste Zeichen jedes Satzes wird als Vorschubsteuerzeichen interpretiert und in das Vorschubsteuerzeichen des SC1 umcodiert.

3.2. O-Format

Die Sätze enthalten keine Vorschubsteuerzeichen. Vor jedem Satz wird das Vorschubsteuerzeichen NL des ZC1 eingefügt.

3.3. Anzahl der Schreibstellen

Bei der Ausgabe über Fernschreiber werden maximal 69 Zeichen pro Zeile ausgegeben. Sind mehr als 69 Zeichen pro Zeile vorhanden, so werden automatisch nach dem 69. Zeichen jeder Zeile die Steuerzeichen CR CR LF (SC1-Code) eingefügt.

4. Zuordnung

4.1. Vorschubsteuerzeichen

Im A-Format wird das erste Zeichen eines Satzes stets als Steuerzeichen für den Zeilenvorschub interpretiert. Nur die rechten 4 Bits des Zeichens werden interpretiert, die linken 4 Bits werden dagegen so interpretiert, als ob sie 000L enthalten. Bedeutung der ZC1-Vorschubsteuerzeichen:

ZC1		SC1
Zeichen	dez.	Zeichen
NL6	16	CR CR LF LF LF LF LF LF
NL5	17	CR CR LF LF LF LF LF
NL4	18	CR CR LF LF LF LF
NL3	19	CR CR LF LF LF
NL2	20	CR CR LF LF
NL	21	CR CR LF
CR	22	CR CR
NF	23	} CR CR LF
VT	24	
VT3	25	
VT4	26	
VT5	27	
VT6	28	
VT7	29	}
VT8	30	
NL7	31	CR CR LF LF LF LF LF LF LF LF

CR ≙ 02 }
 LF ≙ 08 } im SC1

Frühere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Änderungen:

N3/NM

N31/V23

Böckmann

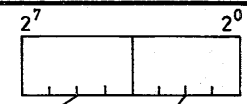
Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2 - 4

V

Code - Tabelle



0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	L0LO	L0LL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0	16	32	48	64	80	96	112	128	144 Zi 11	160 Zi 1E	176 Zi 0D	192 Bu 18	208 Bu 1D	224 Bu 18	240 Bu 1D	0	0000
1	17	33	49	65	81	97 Zi 14	113	129	145 Zi 18	161 Zi 09	177 Zi 1D	193 Bu 13	209 Bu 0A	225 Bu 13	241 Bu 0A	1	000L
2	18	34	50	66	82	98	114	130	146	162 Zi 16	178 Zi 19	194 Bu 0E	210 Bu 14	226 Bu 0E	242 Bu 14	2	00LO
3	19	35	51	67	83	99	115	131	147 Zi 17	163 Zi 0B	179 Zi 10	195 Bu 12	211 Bu 01	227 Bu 12	243 Bu 01	3	00LL
4	20	36	52	68	84	100	116	132	148	164	180 Zi 0A	196 Bu 10	212 Bu 1C	228 Bu 10	244 Bu 1C	4	0L00
5	21	37	53 FL	69	85	101	117	133	149	165	181 Zi 01	197 Bu 16	213 Bu 0F	229 Bu 16	245 Bu 0F	5	0LOL
6	22	38	54	70	86	102	118	134	150	166	182 Zi 15	198 Bu 0B	214 Bu 19	230 Bu 0B	246 Bu 19	6	0LLO
7	23	39	55	71	87	103	119 Bu 00	135	151 Zi 0F	167	183 Zi 1C	199 Bu 05	215 Bu 17	231 Bu 05	247 Bu 17	7	0LLL
8	24	40	56	72	88	104	120 Zi 13	136	152	168	184 Zi 0C	200 Bu 0C	216 Bu 15	232 Bu 0C	248 Bu 15	8	L000
9	25	41	57	73	89	105	121	137	153	169 Zi 07	185 Zi 03	201 Bu 1A	217 Bu 11	233 Bu 1A	249 Bu 11	9	L00L
10	26	42	58	74	90	106	122	138	154	170 Zi 06	186	202 Bu 1E	218	234 Bu 1E	250	A	L0LO
11	27	43	59	75	91	107	123	139	155	171 Zi 0E	187	203 Bu 09	219	235 Bu 09	251	B	L0LL
12	28	44	60	76	92	108	124 Zi 00	140	156	172 Zi 1A	188	204 Bu 07	220	236 Bu 07	252	C	LL00
13	29	45	61	77	93	109	125	141 Bu 10	157	173 Zi 05	189	205 Bu 06	221	237 Bu 06	253	D	LL0L
14	30	46	62	78	94	110	126	142	158	174	190	206 Bu 03	222	238 Bu 03	254	E	LLLO
15	31	47	63	79	95	111	127	143	159	175 Bu 04	191	207 Bu 0D	223	239 Bu 0D	255	F	LLLL

wird durch Ersatzzeichen dargestellt


wird Zi 05 zugeordnet

Übersichtstabelle

2⁷ 2⁰
 1. Sede- zimale 2. Sede- zimale

0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	L0LL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0 NUL	16 NL6	32 SUB	48	64	80	96 "	112 %	128 ^	144 +	160 (176 0	192 A	208 Q	224 a	240 q	0	0000
1 NU1	17 NL5	33 EM	49	65	81	97 ,	113 §	129 v	145 -	161)	177 1	193 B	209 R	225 b	241 r	1	000L
2 NU2	18 NL4	34	50	66	82	98 ,	114 #	130 -	146	162 [178 2	194 C	210 S	226 c	242 s	2	00LO
3 NU3	19 NL3	35	51	67	83	99 ,	115 \$	131 /	147	163]	179 3	195 D	211 T	227 d	243 t	3	00LL
4 NU4	20 NL2	36	52	68	84	100	116 ¢	132	148	164 {	180 4	196 E	212 U	228 e	244 u	4	0LOO
5 NU5	21 NL	37 TE	53 FL	69	85	101 ^	117	133	149	165 }	181 5	197 F	213 V	229 f	245 v	5	0LOL
6	22 CR	38	54	70	86	102 °	118 @	134 ↑	150	166 (182 6	198 G	214 W	230 g	246 w	6	0LLO
7	23 NF	39	55	71	87	103 -	119 &	135	151 =	167)	183 7	199 H	215 X	231 h	247 x	7	0LLL
8	24 VT	40	56	72	88	104	120 *	136	152 ≠	168 MZ	184 8	200 I	216 Y	232 i	248 y	8	L000
9	25 VT3	41	57	73	89	105	121	137	153	169 .	185 9	201 J	217 Z	233 j	249 z	9	L00L
10	26 VT4	42	58	74	90	106	122	138	154	170 ,	186	202 K	218 Ä	234 k	250 ä	A	LOLO
11	27 VT5	43	59	75	91	107 /	123	139	155	171 :	187	203 L	219 Ö	235 l	251 ö	B	L0LL
12	28 VT6	44	60	76	92	108 ,	124 □	140	156 v	172 ;	188	204 M	220 Ü	236 m	252 ü	C	LL00
13	29 VT7	45	61	77	93	109 ,	125	141 10:	157 VII	173 !	189	205 N	221	237 n	253 ß	D	LL0L
14	30 VT8	46	62	78	94	110 -	126	142 III	158 IV	174 ?	190	206 O	222	238 o	254	E	LLLO
15	31 NL7	47	63	79	95	111 -	127 π	143	159	175 SP	191 PZ	207 P	223	239 p	255	F	LLLL

 wird durch Ersatzzeichen dargestellt

 nicht erlaubte Zeichen; wird durch ! dargestellt

6R 2a

4.2. Zeichen

Die Zeichen werden gemäß der Code-Tabelle zugeordnet. Die Zeichen, denen in der Code-Tabelle keine Lochung zugeordnet ist, werden auf das Zeichen ! (Zi 05) zugeordnet. Für die Zeichen, deren Felder punktiert sind, wird ein Ersatzzeichen ausgegeben. Es sind dies:

ZC1		SC1	
Zeichen	dez.	Zeichen	Lochg
¹⁰ Basiszehn	141	' Apostroph	Zi 14
Kleinbuchstaben		Großbuchstaben	
Alle nicht darstellbaren Zeichen		! Ausrufezeichen	Zi 05
NUL N11 NU1...NU5	0 1..5	Keine Zuordnung	
FL Fluchtsymbol	53	Keine Lochung 5x Durchlochung	00 1F

Den Steuerzeichen NUL, NU1 bis NU5 wird kein Ausgabezeichen zugeordnet; es tritt also entsprechend seiner Bedeutung im Ausgabebetext nicht als Zeichen auf.

5. Sonstiges

Bei der Ausgabe einer Datei werden in der nachstehenden Reihenfolge ausgegeben:

Klarschriftvorspann
 Leerband
 Vorspann
 Information
 Nachspann
 Leerband

5.1. Klarschriftvorspann

Der Klarschriftvorspann enthält ein Lochmuster, das Klarschrift darstellt und vom Bedienungspersonal für die Identifizierung gelesen werden kann.

5.2. Leerband

Das Leerband besteht nur aus Transportlöchern und hat eine Länge von 400 Zeichen (~ 1 m).

5.3. Vorspann, Nachspann

Vor- und Nachspann bestehen aus jeweils 100 Zeichen ZI.



Informationsverarbeitung
Lochstreifencode

SC2

2N 0813

.212

1. Zweck und Anwendung

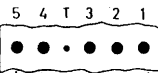
Der Lochstreifencode SC2 entspricht dem in DIN 66 006 aufgeführten Lochstreifencode. Er ist speziell für die Darstellung von ALGOL-Symbolen auf 5-Spur-Lochstreifen gedacht.

2. Darstellung auf Lochstreifen

Für die Darstellung des Codes werden Lochstreifen nach DIN 66 016 verwendet. Der Code wird in den Spuren 1 bis 5 dargestellt. Für die binäre Darstellung wird der Spur 1 das Bit mit der Wertigkeit 2^0 zugeordnet.

3. Code-Tabelle

Bu		Zi		
0	1	0	1	
	E		3	0
T	Z	5	+	1
CR	D	CR	⊕	2
O	B	9	×	3
SP	S	SP	'	4
H	Y	10	6	5
N	F	,	[6
M	X	.	/	7
LF	A	LF	-	8
L	W)	2	9
R	J	4	;	A
G	ZI]	ZI	B
I	U	8	7	C
P	Q	0	1	D
C	K	:	(E
V	BU	=	BU	F



4. Steuerzeichen

CR Wagenrücklauf (Carriage Return)

Das Zeichen dient zur Rückbewegung der Schreibeinrichtung auf die erste Schreibposition der gleichen Zeile.

LF Zeilenvorschub (Line Feed)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung auf die gleiche Schreibposition der nächsten Zeile.

BU Buchstabenumschaltung

Alle folgenden Zeichen sind nach der Buchstabenseite (Bu) der Code-Tabelle zu interpretieren.

ZI Ziffernumschaltung

Alle folgenden Zeichen sind nach der Ziffernseite (Zi) der Code-Tabelle zu interpretieren.

SP Zwischenraum (Space)

Das Zeichen dient zur Vorwärtsbewegung der Schreibeinrichtung um eine Schreibposition.

⊕ Wer da?

Das Zeichen löst den fremden Namengeber aus.

Erläuterungen

Ausgangspunkt für diesen Code ist das internationale Telegraphen-Alphabet Nr. 2 der CCITT und die Darstellung von ALGOL-Symbolen auf Lochstreifen nach DIN 66 006. In den folgenden Punkten wurde davon abgewichen:

SC2	CCITT 2
x	?
;	⊕
[
]	
10	

Fehlere Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig missbräuchlich benutzt werden

Änderungen:

N3/N

N 31/V 23

Ersatz für:

Ersetzt durch:

V



Informationsverarbeitung
Umcodierleistung Lochstreifeneingabe TR 440
 SC2 → ZC1

2N 0815

.412

Einsprüche bis 31. August 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3: C o d e s , Schriftführer: Böckmann N31/V23, Tel. 2474

1. Zweck und Anwendung

Diese Werknorm legt fest, welche Umcodierleistung bei

Lochstreifeneingabe über Streifenleser,
 Eingabe über die Fernschreibtastatur,
 Eingabe über den Lochstreifenleser des Fernschreibers

erbracht werden.

Den Zeichen des Lochstreifencodes SC2 (Werknorm 2N 0813.212) werden Oktaden des Zentralcodes ZC1 (Werknorm 2N 0812.511) zugeordnet.

Für die vorliegende Norm gelten folgende Kennzeichnungen:

- Symbolischer Name bei der Steuerung der Eingabeprozedur: SC2
- Kennzeichen der Codeeinstellung (s. Abschn. 4.1 und 5.1): BU'03'
- Bezieht sich auf Lochstreifencode (Werknorm 2N 0813.212): SC2

2. Darstellung

In den Feldern der Code-Tabelle ist jeweils oben das Zeichen angegeben, das der Lochkombination des Lochstreifencodes SC2 zugeordnet ist. Im unteren Teil des Feldes ist der dezimale Wert der Oktade angegeben, der diese Lochkombination zugeordnet wird.

Die Zeichen BU und ZI dienen der Umschaltung auf die linke bzw. rechte Hälfte der Code-Tabelle und werden nicht einer Oktade zugeordnet.

BU und ZI sind Steuerzeichen.
 Bu und Zi bezeichnen die Hälften der Tabelle.

Soweit im Text zwei Zeichen von Apostrophen eingeschlossen sind, handelt es sich um die sedezimale Darstellung der Lochkombination.

3. Zuordnung

Die Zuordnung ist der Code-Tabelle zu entnehmen. Eine andere Zuordnung ergibt sich in den im Abschnitt 5 aufgeführten Fällen. Das Zeichen CR wird ignoriert.

Code-Tabelle

Bu		Zi		
0	1	0	1	
	E		3	0
	196		179	
T	Z	5	+	1
211	217	181	144	
CR	D	CR	⊕	2
	195			
O	B	9	×	3
206	193	185	120	
SP	S	SP	'	4
175	210	175	97	
H	Y	6	6	5
199	216	53	182	
N	F	'	[6
205	197	170	162	
M	X	.	/	7
204	215	169	147	
LF	A	LF	-	8
21	192	21	145	
L	W)	2	9
203	214	161	178	
R	J	4	;	A
209	201	180	172	
G	ZI] ZI		B
198		163		
I	U	8	7	C
200	212	184	183	
P	Q	0	1	D
207	208	176	177	
C	K	:	(E
194	202	171	160	
V	BU	=	BU	F
213		151		



keine Zuordnung

4. Fluchtsymbol

4.1. Codeunabhängiges Fluchtsymbol (FU)

Das codeunabhängige Fluchtsymbol besteht aus

- Leerband (mindestens 1 Zeichen)
- 5 Zeichen BU ('1F')

Es folgt stets unmittelbar das Kennzeichen für die Codeeinstellung (s. Abschnitt 1). Das codeunabhängige Fluchtsymbol wird nur zur Einleitung der Kommandos für die Eingabeprozedur verwendet (s. Abschnitt 5.1).

re Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

erzungen:

Rm

N31/N

N31/V23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2

V

4.2. Codeabhängiges Fluchtsymbol (Fa)

Das codeabhängige Fluchtsymbol ist für den vorliegenden Code das Zeichen

10 (ZI'05')

Mehrere aufeinanderfolgende Fluchtsymbole werden als ein Fluchtsymbol behandelt.

5. Spezielle Folgen

In den nachstehenden Fällen wird eine andere Zuordnung vorgenommen als in Abschnitt 3 angegeben.

5.1. Anweisungen an die Eingabeprozedur

Folgt unmittelbar auf ein codeunabhängiges Fluchtsymbol (Fu) mit nachfolgendem Kennzeichen für die Code-einstellung das Zeichen

X (BU'17')

oder auf ein codeabhängiges Fluchtsymbol (Fa) das Zeichen

X (BU'17')
(ZI'1E')

so ist dies eine Anweisung an die Eingabeprozedur. Ein codeabhängiges Fluchtsymbol (Fa) mit nachfolgendem Punkt beendet diese Anweisung.

Die Zuordnung der Lochungen der vorstehenden Anweisungen zu Oktaden wird durch die Eingabeprozedur festgelegt.

5.2. Dezimaler Oktadenwert

Falls die Darstellung einer beliebigen Oktade durch ihren dezimalen Wert in der Eingabeprozedur vorgesehen ist, gilt nachstehende Festlegung.

Folgen einem codeabhängigen Fluchtsymbol drei Ziffern (zugehörige Lochungen siehe Code-Tabelle), so stellt diese dreistellige Dezimalzahl einen Oktadenwert dar. Der Folge (Fluchtsymbol, drei Ziffern) wird der angegebene Oktadenwert zugeordnet.

10	0	9	6
----	---	---	---

 ergibt

96

Es sind nur die Werte von 064 bis 255 zugelassen.

5.3. Vorspann, Nachspann

Bei Lochstreifen muß vor und hinter der Information ein Vorspann bzw. Nachspann vorhanden sein. Er besteht jeweils aus

mindestens 50 Zeichen Leersprossen

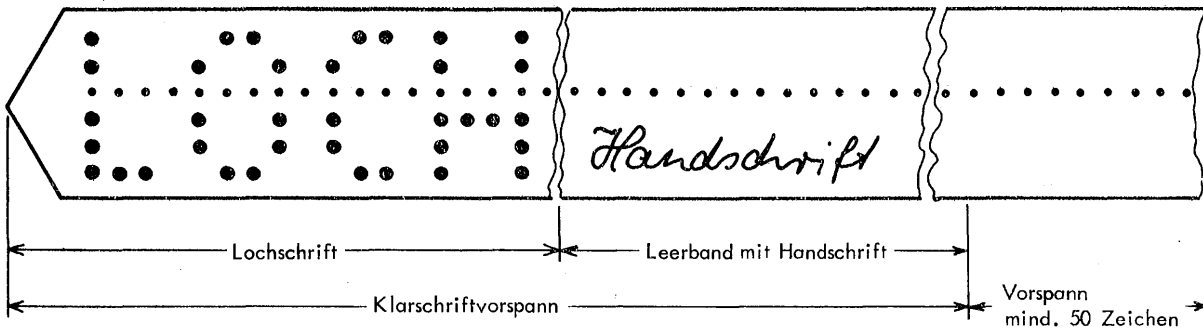
Ein Lochstreifen muß so eingelegt werden, daß die Lesestation innerhalb des Vorspanns zu lesen beginnt.

Bei der Eingabe über den Lochstreifenleser des Fernschreibers muß die Eingabe abgebrochen werden, solange der Nachspann gelesen wird.

Die Leersprosse besteht nur aus dem Transportloch der Taktspur. Bei der Eingabe über die Fernschreibstatur entfällt Vor- und Nachspann.

5.4. Klarschriftvorspann

Der Klarschriftvorspann liegt vor dem Vorspann und kann beliebige Information enthalten. Er darf jedoch nicht mit eingelesen werden.





Informationsverarbeitung
Umcodierleistung Lochstreifenausgabe TR 440
 ZC1 → SC2

2N 0815

.512

Einsprüche bis 31. August 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen feilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an:

Arbeitsgruppe AG3: C o d e s, Schriefführer: Böckmann N31/V23, Tel. 2474

1. Zweck und Anwendung

Diese Werknorm legt fest, welche Umcodierleistung bei der Lochstreifenausgabe im Code SC2 des TR 440 erbracht wird. Den Oktaden des Zentralcodes ZC1 (Werknorm 2 N 0812.511) werden Lochmuster des Lochstreifencodes SC1 (Werknorm 2 N 0813.212) zugeordnet.

2. Bezeichnung

In der Code-Tabelle wird vom Bitmuster des Zentralcodes ZC1 ausgegangen. In der Kopfleiste und der rechten Seitenleiste sind Dualwert und Sedezimalwert des ZC1 und in der linken oberen Ecke eines jeden Feldes der Dezimalwert der ZC1-Oktade angegeben. Die Angabe im Feld verweist auf das zugehörige Zeichen im SC2.

Alle Angaben für das zugehörige Zeichen im Code SC2 bestehen aus einer zweistelligen Sedezimalzahl. Sie stellt das Lochmuster des SC2-Zeichens dar. Ein vorangestelltes "Bu" bedeutet, daß das Zeichen nach der Buchstabenleiste der Code-Tabelle zu interpretieren ist; bei einem vorangestellten "Zi" ist das Zeichen nach der Ziffernleiste zu interpretieren. Ist "Bu" oder "Zi" nicht angegeben, so hat das Zeichen sowohl auf der Buchstaben- als auch auf der Ziffernseite dieselbe Bedeutung.

Die Übersichtstabelle entspricht den Zentralcode ZC1 und gibt eine Übersicht über die Zeichen, die als Ausgabezeichen dargestellt werden können. Die Übersichtstabelle ist nicht Bestandteil dieser Werknorm.

3. Informationsdarstellung

Die auszugebende Information hat die Form einer Datei vom Typ sequentiell. Jede Zeile bildet einen Satz der Datei. Die Information einer Datei kann entweder im A-Format oder im O-Format vorliegen. Das Format bestimmt die Art der Vorschubsteuerung und ist bei der Erteilung des Auftrags mit anzugeben.

3.1. A-Format

Das erste Zeichen jedes Satzes wird als Vorschubsteuerzeichen interpretiert und in das Vorschubsteuerzeichen des SC2 umcodiert.

3.2. O-Format

Die Sätze enthalten keine Vorschubzeichen. Vor jedem Satz wird das Vorschubsteuerzeichen NL des ZC1 eingefügt.

3.3. Anzahl der Schreibstellen

Bei der Ausgabe über Fernschreiber werden maximal 69 Zeichen pro Zeile ausgegeben. Sind mehr als 69 Zeichen pro Zeile vorhanden, so werden automatisch nach dem 69. Zeichen die Steuerzeichen CR CR LF (SC2-Code) eingefügt.

4. Zuordnung

4.1. Vorschubsteuerzeichen

Im A-Format wird das erste Zeichen eines Satzes stets als Steuerzeichen für den Zeilenvorschub interpretiert. Nur die rechten 4 Bits des Zeichens werden interpretiert, die linken 4 Bits werden dagegen so interpretiert, als ob sie 000L enthalten. Bedeutung der ZC1-Vorschubsteuerzeichen:

ZC1		SC2
Zeichen	dez.	Zeichen
NL6	16	CR CR LF LF LF LF LF LF
NL5	17	CR CR LF LF LF LF LF
NL4	18	CR CR LF LF LF LF
NL3	19	CR CR LF LF LF
NL2	20	CR CR LF LF
NL	21	CR CR LF
CR	22	CR CR
NF	23	} CR CR LF
VT	24	
VT3	25	
VT4	26	
VT5	27	
VT6	28	
VT7	29	} im SC2
VT8	30	
NL7	31	CR CR LF LF LF LF LF LF LF LF

CR ≙ 02 }
 LF ≙ 08 } im SC2

N3/N

N31/V23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2 bis 4

V

re Ausgaben:

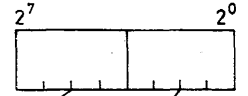
Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

erungen:

Rm

Code - Tabelle

ZC1 → SC2



0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	L0LL	LL00	LL0L	LLLO	LLLL			
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F			
0	16	32	48	64	80	96	112	128	144	Zi 11	Zi 1E	Zi 0D	Bu 18	Bu 1D	Bu 18	Bu 1D	0	0000
1	17	33	49	65	81	Zi 14	97	113	129	Zi 18	Zi 09	Zi 1D	Bu 13	Bu 0A	Bu 13	Bu 0A	1	000L
2	18	34	50	66	82	98	114	130	146	Zi 16	Zi 19	Bu 0E	Bu 14	Bu 0E	Bu 14	2	00LO	
3	19	35	51	67	83	99	115	131	147	Zi 17	Zi 0B	Zi 10	Bu 12	Bu 01	Bu 12	Bu 01	3	00LL
4	20	36	52	68	84	100	116	132	148	164	Zi 0A	Bu 10	Bu 1C	Bu 10	Bu 1C	4	0L00	
5	21	37	53 1x00 5xBu 1F	69	85	101	117	133	149	165	Zi 01	Bu 16	Bu 0F	Bu 16	Bu 0F	5	0LOL	
6	22	38	54	70	86	102	118	134	150	166	Zi 15	Bu 0B	Bu 19	Bu 0B	Bu 19	6	0LLO	
7	23	39	55	71	87	103	119	135	151	Zi 0F	Zi 1C	Bu 05	Bu 17	Bu 05	Bu 17	7	0LLL	
8	24	40	56	72	88	104	Zi 13	120	136	152	168	Zi 0C	Bu 0C	Bu 15	Bu 0C	Bu 15	8	L000
9	25	41	57	73	89	105	121	137	153	Zi 07	Zi 03	Bu 1A	Bu 11	Bu 1A	Bu 11	9	L00L	
10	26	42	58	74	90	106	122	138	154	Zi 06		Bu 1E		Bu 1E		A	LOLO	
11	27	43	59	75	91	107	123	139	155	Zi 0E		Bu 09		Bu 09		B	L0LL	
12	28	44	60	76	92	108	Zi 05	124	140	156	Zi 1A	Bu 07	Bu 07	Bu 07	252	C	LL00	
13	29	45	61	77	93	109	125	Zi 05	141	157	173	Bu 06	Bu 06	Bu 06	253	D	LL0L	
14	30	46	62	78	94	110	126	142	158	174	190	Bu 03	Bu 03	Bu 03	254	E	LLLO	
15	31	47	63	79	95	111	127	143	159	Bu 04	191	Bu 0D	Bu 0D	Bu 0D	255	F	LLLL	

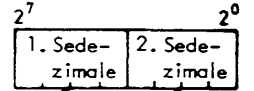
wird durch Ersatzzeichen dargestellt

wird Zi 13 zugeordnet

GR 35

Übersichtstabelle

ZC1→SC2



0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	LOLL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0 NUL	16 NL6	37 SUB	48	59	80	96 #	112 %	128 ^	144 +	160 (176 0	192 A	208 Q	224 a	240 q	0	0000
1 NU1	17 NL5	33 EM	49	65	81	97 ,	113 §	129 v	145 -	161)	177 1	193 B	209 R	225 b	241 r	1	000L
2 NU2	18 NL4	34	50	66	82	98 ,	114 #	130 _	146	162 [178 2	194 C	210 S	226 c	242 s	2	00LO
3 NU3	19 NL3	35	51	67	83	99 ,	115 \$	131 /	147 /	163]	179 3	195 D	211 T	227 d	243 t	3	00LL
4 NU4	20 NL2	36	52	68	84	100	116 f	132	148	164 {	180 4	196 E	212 U	228 e	244 u	4	0LOO
5 NU5	21 NL	37 TE	53 FL	69	85	101 ^	117	133	149	165 }	181 5	197 F	213 V	229 f	245 v	5	0LOL
6	22 CR	38	54	70	86	102 o	118 @	134 †	150	166 <	182 6	198 G	214 W	230 g	246 w	6	0LLO
7	23 NF	39	55	71	87	103 _	119 &	135	151 =	167 >	183 7	199 H	215 X	231 h	247 x	7	0LLL
8	24 VT	40	56	72	88	104	120 *	136	152 ≠	168 MZ	184 8	200 I	216 Y	232 i	248 y	8	L000
9	25 VT3	41	57	73	89	105	121	137	153	169 .	185 9	201 J	217 Z	233 j	249 z	9	L00L
10	26 VT4	42	58	74	90	106	122	138	154	170 ,	186	202 K	218 Ä	234 k	250 ä	A	LOLO
11	27 VT5	43	59	75	91	107 /	123	139	155 v	171 :	187	203 L	219 Ö	235 l	251 ö	B	LOLL
12	28 VT6	44	60	76	92	108 /	124 □	140	156 v	172 ;	188	204 M	220 Ü	236 m	252 ü	C	LLOO
13	29 VT7	45	61	77	93	109 /	125	141 10	157 ≡	173 !	189	205 N	221	237 n	253 ß	D	LLOL
14	30 VT8	46	62	78	94	110 /	126	142	158 ≡	174 ?	190	206 O	222	238 o	254	E	LLLO
15	31 NL7	47	63	79	95	111 /	127 π	143	159	175 SP	191 PZ	207 P	223	239 p	255	F	LLLL

wird durch Ersatzzeichen dargestellt

nicht erlaubte Zeichen; wird durch > dargestellt

GR 2a

4.2. Zeichen

Die Zeichen werden gemäß der Code-Tabelle zugeordnet. Die Zeichen, denen in der Code-Tabelle keine Lochung zugeordnet ist, werden auf das Zeichen \times (Zi 13) zugeordnet. Für die Zeichen, deren Felder punktiert sind, wird ein Ersatzzeichen ausgegeben. Es sind dies:

ZC1		SC2	
Zeichen	dez.	Zeichen	Lochg
Kleinbuchstaben		Großbuchstaben	
Alle nicht darstellbaren Zeichen		\times Malkreuz	Zi 13
NUL Nil NU1...NU5	0 1..5	Keine Zuordnung	
FL Fluchtsymbol	53	Leersprosse 5 x Bu	1x00 5x1F

Den Steuerzeichen NUL, NU1 bis NU5 wird kein Ausgabezeichen zugeordnet; es tritt also entsprechend seiner Bedeutung im Ausgabebetext nicht als Zeichen auf.

5. Sonstiges

Bei der Ausgabe einer Datei werden in der nachstehenden Reihenfolge ausgegeben:

Klarschriftvorspann
Leerband
Information
Leerspann

5.1. Klarschriftvorspann

Der Klarschriftvorspann enthält ein Lochmuster, das Klarschrift darstellt und vom Bedienungspersonal für die Identifizierung gelesen werden kann.

5.2. Leerband

Das Leerband besteht nur aus Transportlöchern und hat eine Länge von 400 Zeichen (~1 m).



Informationsverarbeitung
Druckercode

DC1

2N 0813

.311

1. Zweck und Anwendung

Der Druckercode DC1 bestimmt die Steuerung des Schnell-druckers und gibt die Zuordnung zwischen Bitmuster und auszudruckendem Zeichen an.

Der Zeichenvorrat ist identisch mit dem des Zeichensatzes ZS 64 (siehe Werknorm 2N 0810. 111). ¹⁾

Es wird die Form der Schriftzeichen festgelegt.

2. Code-Tabelle

Die 7-Bit-Zeichen des Codes werden durch ein 8. Bit so ergänzt, daß eine ungerade Anzahl auf L gesetzter Bits entsteht. In der Code-Tabelle (mit Paritybit) sind alle nicht erlaubten Bitkombinationen gekennzeichnet.

In der Code-Tabelle und in der Übersichtstabelle ist der größte Teil der Zeichen doppelt vorhanden, den nicht gekennzeichneten ist der Vorzug zu geben.

Die Übersichtstabelle ist eine Darstellung ohne Paritybit. Sie dient nur der Übersicht. Auf sie wird nicht Bezug genommen.

3. Druckersteuerung

Von jeder Zeichenfolge, die dem Drucker übergeben wird, ist das erste Zeichen ein Vorschubsteuerzeichen. Die Bedeutung dieses Steuerzeichens ist den Anhängen zu dieser Norm zu entnehmen. Die weiteren Zeichen stellen mit Ausnahme der Zeichen NUL und ZE die Druckinformation einer Zeile dar. Werden mehr Zeichen angeliefert, als Schreibstellen in der Zeile vorhanden sind, so werden die überzähligen Zeichen wie das Steuerzeichen NUL behandelt.

Die in der Code-Tabelle angegebenen Steuerzeichen haben folgende Bedeutung:

NUL Nil (Null)

Dieses Zeichen ist ein Füllzeichen. Es wird vom Drucker ignoriert, tritt also im Druckbild nicht in Erscheinung.

SP Zwischenraum (Space)

Dieses Zeichen bewirkt, daß die zugehörige Druckposition unbedruckt bleibt.

ZE Zeilenende

Dieses Zeichen zeigt an, daß für diese Zeile alle Zeichen übergeben sind. Alle folgenden Zeichen werden ignoriert.

4. Zeichenanordnung auf der Druckwalze

Die Zeichen sind in der folgenden Reihenfolge auf dem Umfang der Druckwalze angeordnet:

Reihe Zeichen Code

1	0	176
2	1	49
3	2	50
4	3	179
5	4	52
6	5	181
7	6	182
8	7	55
9	8	56
10	9	185
11	,	44
12	-	173
13	□ ¹⁾	35
14	.	174
15	A	193
16	B	194
17	C	67
18	D	196
19	E	69
20	F	70
21	G	199
22	H	200
23	I	73
24	J	74
25	K	203
26	L	76
27	M	205
28	N	206
29	O	79
30	P	208
31	Q	81
32	R	82

Reihe Zeichen Code

33	S	211
34	T	84
35	U	213
36	V	214
37	W	87
38	X	88
39	Y	218
40	Z	219
41	+	171
42	¬ ¹⁾	162
43	*	42
44	/	47
45	=	61
46	(40
47)	41
48	:	186
49	;	59
50	'	167
51	[164
52	_	223
53]	64
54	!	161
55	?	191
56	<	188
57	>	62
58	& ¹⁾	38
59	%	37
60		94
61	Ä	91
62	Ö	220
63	Ü	93
64	.	174

¹⁾ siehe Erläuterungen

N3/N

N 31/V23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2 bis 5

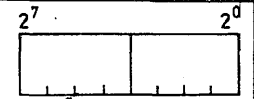
Frühere Ausgaben: Dezember 69

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Anmerkungen: im Anhang I wurde behoben

hm

Code - Tabelle (mit Paritybit)



0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	L0LO	L0LL	LL00	LL0L	LLLO	LLLL									
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F									
0	16	32	SP	48	64]	80	96	112	P	128	NUL	144	160	176	0	192	208	P	224]	240	0	0000
1	17	33	1	49	65	Q	81	97	A	113	129	145	161	177	!	193	A	209	225	!	241	Q	1	000L
2	18	34	2	50	66	R	82	98	B	114	130	146	162	178	—	194	B	210	226	—	242	R	2	00LO
3	19	35	☐	51	67	C	83	99	S	115	131	☐	147	163	3	195	S	211	227	☐	243	C	3	00LL
4	20	36	4	52	68	T	84	100	D	116	132	148	164	180	[196	D	212	228	—	244	T	4	0LOO
5	21	37	%	53	69	E	85	101	U	117	133	%	149	165	5	197	U	213	229	—	245	E	5	0LOL
6	22	38	&	54	70	F	86	102	V	118	134	&	150	166	6	198	V	214	230	—	246	F	6	0LLO
7	23	39	7	55	71	W	87	103	G	119	135	151	167	/	183	G	199	215	231	—	247	W	7	0LLL
8	24	40	8	56	72	X	88	104	H	120	136	152	168	(184	H	200	216	232	—	248	X	8	L000
9	25	41)	57	73	I	89	105	Y	121	137)	153	169	9	201	Y	217	233	—	249	I	9	L00L
10	26	42	*	58	74	J	90	106	Z	122	138	*	154	170	:	202	Z	218	234	—	250	J	A	L0LO
11	27	43	;	59	75	Ä	91	107	K	123	139	;	155	171	+	203	K	219	235	—	251	Ä	B	L0LL
12	28	44	<	60	76	L	92	108	Ö	124	140	<	156	172	<	204	Ö	220	236	—	252	L	C	LL00
13	29	45	=	61	77	Ü	93	109	M	125	141	=	157	173	-	205	M	221	237	—	253	Ü	D	LL0L
14	30	46	>	62	78	I	94	110	N	126	142	>	158	174	.	206	N	222	238	—	254	I	E	LLLO
15	31	47	?	63	79	O	95	111	—	127	143	/	159	175	?	207	—	223	239	—	255	O	F	LLLL

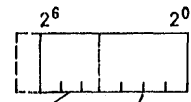
nicht erlaubte Bitkombination

mehrfach im Code vorhanden

1) siehe Erläuterungen

7749

Übersichtstabelle



000	00L	0LO	0LL	L00	L0L	LLO	LLL		
0	1	2	3	4	5	6	7		
⁰ NUL	¹⁶ -	³² SP	⁴⁸ 0	⁶⁴]	⁸⁰ P	⁹⁶ I	¹¹² P	0	0000
¹	¹⁷ -	³³ !	⁴⁹ 1	⁶⁵ A	⁸¹ Q	⁹⁷ A	¹¹³ O	1	000L
² -	¹⁸ -	³⁴ -	⁵⁰ 2	⁶⁶ B	⁸² R	⁹⁸ B	¹¹⁴ R	2	00LO
³ H	¹⁹ -	³⁵ ▣	⁵¹ 3	⁶⁷ C	⁸³ S	⁹⁹ C	¹¹⁵ S	3	00LL
⁴ [²⁰ -	³⁶ [⁵² 4	⁶⁸ D	⁸⁴ T	¹⁰⁰ D	¹¹⁶ T	4	0L00
⁵ %	²¹ -	³⁷ %	⁵³ 5	⁶⁹ E	⁸⁵ U	¹⁰¹ E	¹¹⁷ U	5	0L0L
⁶ &	²² -	³⁸ &	⁵⁴ 6	⁷⁰ F	⁸⁶ V	¹⁰² F	¹¹⁸ V	6	0LLO
⁷ ;	²³ -	³⁹ /	⁵⁵ 7	⁷¹ G	⁸⁷ W	¹⁰³ G	¹¹⁹ W	7	0LLL
⁸ (²⁴ -	⁴⁰ (⁵⁶ 8	⁷² H	⁸⁸ X	¹⁰⁴ H	¹²⁰ X	8	L000
⁹)	²⁵ -	⁴¹)	⁵⁷ 9	⁷³ I	⁸⁹ Y	¹⁰⁵ I	¹²¹ Y	9	L00L
¹⁰ *	²⁶ -	⁴² *	⁵⁸ :	⁷⁴ J	⁹⁰ Z	¹⁰⁶ J	¹²² Z	A	L0LO
¹¹ +	²⁷ +	⁴³ +	⁵⁹ ;	⁷⁵ K	⁹¹ Ä	¹⁰⁷ K	¹²³ Ä	B	L0LL
¹² <	²⁸ <	⁴⁴ ,	⁶⁰ <	⁷⁶ L	⁹² Ö	¹⁰⁸ L	¹²⁴ Ö	C	LL00
¹³ ZE	²⁹ =	⁴⁵ -	⁶¹ =	⁷⁷ M	⁹³ Ü	¹⁰⁹ M	¹²⁵ Ü	D	LL0L
¹⁴ .	³⁰ >	⁴⁶ .	⁶² >	⁷⁸ N	⁹⁴ I	¹¹⁰ N	¹²⁶ I	E	LLLO
¹⁵ /	³¹ ?	⁴⁷ /	⁶³ ?	⁷⁹ O	⁹⁵ -	¹¹¹ O	¹²⁷ -	F	LLLL

mehrfach im Code vorhanden

1) siehe Erläuterungen

GR 39

5. Form der Schriftzeichen

Die Ziffern unterhalb der Zeichen geben den Dezimalwert der Zeichen innerhalb der Codetabelle an



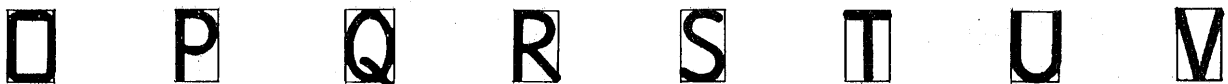
176 49 50 179 52 181 182 55



56 185 193 194 67 196 69 70



199 200 73 74 203 76 205 206



79 208 81 82 211 84 213 214



87 88 217 218 32 171 173 42



47 61 168 41 174 44 167 161



164 64 186 59 38 35 37 188



62 223 162 94 191 91 220 93

GR 155

1) siehe Erläuterungen

Erläuterungen

Die Zeichen, die in der Code-Tabelle nicht gekennzeichnet sind, stimmen mit denen des Druckercode DC2 überein. Die Zeichen, die als mehrfach vorhanden gekennzeichnet sind, haben im Druckercode DC2 andere Zeichenzuordnung.

Bei den ersten Druckerausführungen wurden noch von diesem Code abweichende Zeichen verwendet:

dezimal mit Paritybit	Zeichen
162	— statt —
35	# statt #
38	& statt &

Bei den ersten Druckerausführungen weicht die Form der Schriftzeichen etwas von der im Abschnitt 5 angegebenen Form ab.

Anhang 1

Für den Anelex-Drucker (SDR 176-1) hat das Vorschubsteuerzeichen (siehe Abschnitt 3) folgenden Aufbau:

Parity-bit	a	0	0	0	b
------------	---	---	---	---	---

Ist das Bit a = 0, so wird gemäß dem Vorschublochstreifen vorgeschoben. Der Wert b + 1 gibt die Spur-Nummer (1 bis 8) an.

Ist das Bit a = L, so wird das Papier um b Zeilen vorgeschoben.

Bezüglich des Paritybits siehe Abschnitt 2.



Informationsverarbeitung
Umcodierleistung Druckerausgabe TR 440
ZC 1 → DC 1

2N 0815

.311

Einsprüche bis 31. August 1971

Dieser 2. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwasige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG 3; Codes, Schriftführer: Böckmann N31/V23, Tel. 2474

1. Zweck und Anwendung

Diese Werknorm legt fest, welche Umcodierleistung bei der Druckerausgabe des TR 440 erbracht wird. Den Oktaden des Zentralcodes ZC1 (Werknorm 2N 0812.511) werden Oktaden des Druckercode DC1 (Werknorm 2N 0813.311) zugeordnet.

2. Bezeichnung

In der Code-Tabelle wird vom Bitmuster des Zentralcodes ZC1 ausgegangen. In der Kopfleiste und der rechten Seitenleiste sind Dualwert und Sedezimalwerte des ZC1 und in der linken oberen Ecke eines jeden Feldes der Dezimalwert der ZC1-Oktade angegeben. Die weiter im Feld vorhandene Dezimalzahl gibt den Wert der Oktade im Druckercode DC1 an, die der ZC1-Oktade zugeordnet wird.

Die Übersichtstabelle entspricht dem Zentralcode ZC1 und gibt eine Übersicht über die Zeichen, die auf dem Drucker gemäß dieser Werknorm dargestellt werden können. Sie ist nicht Bestandteil dieser Werknorm.

3. Informationsdarstellung

Die auszugebende Information hat die Form einer Datei vom Typ sequentiell. Jeder Satz der Datei bildet eine Druckzeile. Die Information einer Datei kann entweder im A-Format oder im O-Format vorliegen. Das Format bestimmt die Art der Vorschubsteuerung und ist bei der Erteilung des Druckauftrages mit anzugeben.

3.1. A-Format

Das erste Zeichen jedes Satzes wird als Vorschubsteuerzeichen interpretiert und in das Vorschubsteuerzeichen des DC1 umcodiert.

3.2. O-Format

Die Sätze enthalten keine Vorschubsteuerzeichen. Vor jedem Satz wird das Vorschubsteuerzeichen VT des ZC1 eingefügt.

3.3. Anzahl der Schreibstellen

Alle Druckzeichen eines Satzes werden an den Drucker übergeben. Es wird nicht geprüft, ob die Anzahl der Zeichen je Zeile mit der Papierbreite verträglich ist.

(Werden mehr Druckzeichen an den Drucker übergeben als Druckstellen je Zeile vorhanden sind, so werden die überzähligen Zeichen vom Drucker wie das Steuerzeichen NUL behandelt. Siehe dazu Werknorm der Druckercode.)

4. Zuordnung

4.1. Vorschubsteuerzeichen

Im A-Format wird das erste Zeichen eines Satzes stets als Steuerzeichen für den Zeilenvorschub interpretiert. Nur die rechten 4 Bits des Zeichens werden interpretiert, die linken 4 Bits werden dagegen so interpretiert, als ob sie 000L enthalten. Bedeutung der ZC1-Vorschubsteuerzeichen:

Dez.	Zeichen	Bedeutung
22	CR	kein Vorschub
21	NL	Vorschub um 1 Zeile
20	NL2	Vorschub um 2, 3, ... bzw. 7 Zeilen
19	NL3	
18	NL4	
17	NL5	
16	NL6	
31	NL7	
23	NF	Vorschub nach Kanal 1, Formularvorschub
24	VT	" " " 2
25	VT3	Vorschub nach Kanal 3, 4, ... bzw. 8
26	VT4	
27	VT5	
28	VT6	
29	VT7	
30	VT8	

4.2. Zeichen

Die Zeichen werden gemäß der Code-Tabelle zugeordnet. Für die Zeichen, deren Felder punktiert sind, wird ein Ersatzzeichen ausgegeben. Es sind dies:

ZC1		DC1	
Zeichen	dez.	Zeichen	dez.
TE Textende	37	ZE Zeilenende	13
FL Fluchtsymbol	53	□ Kissen	35
10 Basiszehn	141	' Apostroph	167
< kleiner als	155	< spitze Klammer auf	188
> größer als	156	> spitze Klammer zu	62
MZ Minus Null *	168	¬ Negation	162
PZ Plus Null *	191	& kommerz. Und	38
Kleinbuchstaben		Großbuchstaben	
# Nr.-Zeich.	114	Ä Umlaut groß	91
@ kommerz. a	118	Ö Umlaut groß	220
§ Dollar	115	Ü Umlaut groß	93
" Anführungszeichen	96	[eckige Klamm. auf	164
¢ Cent	116] eckige Klamm. zu	64
Alle nicht darstellbaren ZC1-Zeichen		! Ausrufezeichen	161

* s. unter Erläuterungen

Fertige Ausgaben:

Diese Unterlage darf weder kopiert, noch an drittm Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

Anmerkungen:

Rm

N3/N

N 31/V23

Ersatz für:

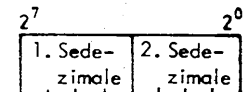
Ersetzt durch:



Fortsetzung Seite 2 - 4

V

Übersichts-Tabelle

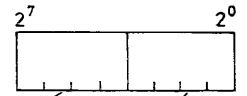
0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	LOOO	LOOL	LOLO	LOLL	LLOO	LLOL	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
⁰ NUL	¹⁶ NL6	³² SUB	⁴⁸ BEL	⁶⁴	⁸⁰	⁹⁶ "	¹¹² %	¹²⁸ ^	¹⁴⁴ +	¹⁶⁰ (¹⁷⁶ 0	¹⁹² A	²⁰⁸ Q	²²⁴ a	²⁴⁰ q	0	0000
¹ NU1	¹⁷ NL5	³³ EM	⁴⁹ DC1	⁶⁵	⁸¹	⁹⁷ '	¹¹³ \$	¹²⁹ v	¹⁴⁵ -	¹⁶¹)	¹⁷⁷ 1	¹⁹³ B	²⁰⁹ R	²²⁵ b	²⁴¹ r	1	000L
² NU2	¹⁸ NL4	³⁴ CAN	⁵⁰ DC2	⁶⁶	⁸²	⁹⁸ /	¹¹⁴ #	¹³⁰ ~	¹⁴⁶	¹⁶² [¹⁷⁸ 2	¹⁹⁴ C	²¹⁰ S	²²⁶ c	²⁴² s	2	00LO
³ NU3	¹⁹ NL3	³⁵	⁵¹ DC3	⁶⁷	⁸³	⁹⁹ \	¹¹⁵ \$	¹³¹	¹⁴⁷ /	¹⁶³]	¹⁷⁹ 3	¹⁹⁵ D	²¹¹ T	²²⁷ d	²⁴³ t	3	00LL
⁴ NU4	²⁰ NL2	³⁶	⁵² DC4	⁶⁸	⁸⁴	¹⁰⁰	¹¹⁶ €	¹³²	¹⁴⁸	¹⁶⁴	¹⁸⁰ 4	¹⁹⁶ E	²¹² U	²²⁸ e	²⁴⁴ u	4	0LOO
⁵ NU5	²¹ NL	³⁷ TE	⁵³ FL	⁶⁹	⁸⁵	¹⁰¹ ^	¹¹⁷	¹³³	¹⁴⁹	¹⁶⁵	¹⁸¹ 5	¹⁹⁷ F	²¹³ V	²²⁹ f	²⁴⁵ y	5	0LOL
⁶ SOH	²² CR	³⁸	⁵⁴	⁷⁰	⁸⁶	¹⁰² o	¹¹⁸ @	¹³⁴ ↑	¹⁵⁰	¹⁶⁶ <	¹⁸² 6	¹⁹⁸ G	²¹⁴ W	²³⁰ g	²⁴⁶ w	6	0LLO
⁷ STX	²³ NF	³⁹	⁵⁵	⁷¹	⁸⁷	¹⁰³ -	¹¹⁹ &	¹³⁵	¹⁵¹ =	¹⁶⁷ >	¹⁸³ 7	¹⁹⁹ H	²¹⁵ X	²³¹ h	²⁴⁷ x	7	0LLL
⁸ ETX	²⁴ VT	⁴⁰	⁵⁶	⁷²	⁸⁸	¹⁰⁴ *	¹²⁰ *	¹³⁶ *	¹⁵² +	¹⁶⁸ {	¹⁸⁴ 8	²⁰⁰ I	²¹⁶ Y	²³² i	²⁴⁸ y	8	L000
⁹ EOT	²⁵ VT3	⁴¹	⁵⁷	⁷³	⁸⁹	¹⁰⁵	¹²¹	¹³⁷	¹⁵³ .	¹⁶⁹	¹⁸⁵ 9	²⁰¹ J	²¹⁷ Z	²³³ j	²⁴⁹ z	9	L00L
¹⁰ ENO	²⁶ VT4	⁴²	⁵⁸	⁷⁴	⁹⁰	¹⁰⁶	¹²²	¹³⁸	¹⁵⁴ ,	¹⁷⁰	¹⁸⁶	²⁰² K	²¹⁸ Ä	²³⁴ k	²⁵⁰ ä	A	LOLO
¹¹ ACK	²⁷ VT5	⁴³ HT	⁵⁹	⁷⁵	⁹¹	¹⁰⁷ \	¹²³	¹³⁹	¹⁵⁵ >	¹⁷¹ :	¹⁸⁷	²⁰³ L	²¹⁹ Ö	²³⁵ l	²⁵¹ ö	B	LOLL
¹² DLE	²⁸ VT6	⁴⁴ BS	⁶⁰ IS4	⁷⁶	⁹²	¹⁰⁸ /	¹²⁴ □	¹⁴⁰ v	¹⁵⁶ v	¹⁷² ;	¹⁸⁸	²⁰⁴ M	²²⁰ Ü	²³⁶ m	²⁵² ü	C	LLOO
¹³ NAK	²⁹ VT7	⁴⁵ ESC	⁶¹ IS3	⁷⁷	⁹³	¹⁰⁹ \	¹²⁵	¹⁴¹ 10	¹⁵⁷ III	¹⁷³ !	¹⁸⁹	²⁰⁵ N	²²¹	²³⁷ n	²⁵³ ß	D	LLOL
¹⁴ SYN	³⁰ VT8	⁴⁶ SO	⁶² IS2	⁷⁸	⁹⁴	¹¹⁰ -	¹²⁶	¹⁴² III	¹⁵⁸ III	¹⁷⁴ ?	¹⁹⁰	²⁰⁶ O	²²²	²³⁸ o	²⁵⁴	E	LLLO
¹⁵ ETB	³¹ NL7	⁴⁷ SI	⁶³ IS1	⁷⁹	⁹⁵	¹¹¹ -	¹²⁷ TE	¹⁴³	¹⁵⁹	¹⁷⁵ SP	¹⁹¹ {	²⁰⁷ P	²²³	²³⁹ p	²⁵⁵ DEL	F	LLLL



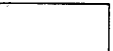
 nicht erlaubtes Zeichen; wird durch ! dargestellt
  wird durch Ersatzzeichen dargestellt
 * s. unter Erläuterungen
 GR 318

Code-Tabelle

0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	LOLO	L0LL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
128	16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	0	0000
128	17	33	49	65	81	97	113	129	145	161	177	193	209	225	241	1	000L
128	18	34	50	66	82	98	114	130	146	162	178	194	210	226	242	2	00LO
128	19	35	51	67	83	99	115	131	147	163	179	195	211	227	243	3	00LL
128	20	36	52	68	84	100	116	132	148	164	180	196	212	228	244	4	0LOO
128	21	37	53	69	85	101	117	133	149	165	181	197	213	229	245	5	0LOL
	22	38	54	70	86	102	118	134	150	166	182	198	214	230	246	6	0LLO
	23	39	55	71	87	103	119	135	151	167	183	199	215	231	247	7	0LLL
	24	40	56	72	88	104	120	136	152	168	184	200	216	232	248	8	L000
	25	41	57	73	89	105	121	137	153	169	185	201	217	233	249	9	L00L
	26	42	58	74	90	106	122	138	154	170	186	202	218	234	250	A	L0LO
	27	43	59	75	91	107	123	139	155	171	187	203	219	235	251	B	L0LL
	28	44	60	76	92	108	124	140	156	172	188	204	220	236	252	C	LL00
	29	45	61	77	93	109	125	141	157	173	189	205	221	237	253	D	LL0L
	30	46	62	78	94	110	126	142	158	174	190	206	222	238	254	E	LLLO
	31	47	63	79	95	111	127	143	159	175	191	207	223	239	255	F	LLLL



 wird durch Ersatzzeichen dargestellt

 wird 161 zugeordnet

* s. unter Erläuterungen

GR 35

4.3. Standardlochung im Vorschubstreifen

Standardmäßig werden folgende Lochungen vorgeschrieben:

Kanal	Loch für die Zeile
1	1
2	1 bis 64
3	
4	
5	Kanal 3 bis 8 eigene Festlegung,
6	aber mindestens
7	1 Loch
8	

Der Standard-Vorschubstreifen ist für Formulare mit 72 Zeilen/Formular vorgesehen.

4.4. Steuerzeichen NUL

Die Steuerzeichen NUL und NU1 bis NU5 werden dem Steuerzeichen NUL im DCJ zugeordnet.

4.5. Steuerzeichen TE

Das Steuerzeichen TE wird dem Steuerzeichen ZE des DCJ zugeordnet. Alle folgenden Zeichen derselben Zeile werden ignoriert.

Erläuterungen

* Die Zeichen PZ und MZ werden mit Rücksicht auf COBOL den Zeichen & bzw. ¬ zugeordnet. Daraus ergibt sich gleichzeitig eine Zuordnung der geschweiften Klammern.



Informationsverarbeitung
Druckercode
DC 2

2N 0813

.312

1. Zweck und Anwendung

Der Druckercode DC 2 bestimmt die Steuerung des Schnell-druckers und gibt die Zuordnung zwischen Bitmuster und auszudruckenden Zeichen an.

Der Zeichenvorrat ist identisch mit dem des Zeichensatzes ZS 115 (siehe Werknorm 2N 0810.111). Die Zeichen 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, , , - , . und E sind auf der Druckwalze zweifach vorhanden. Bei vorzugsweise numerischen Texten wird dadurch die Druckgeschwindigkeit etwa verdoppelt.

Es wird die Form der Schriftzeichen festgelegt.

2. Code-Tabelle

Die 7-Bit-Zeichen des Codes werden durch ein 8. Bit so ergänzt, daß eine ungerade Anzahl auf L gesetzter Bits entsteht. In der Code-Tabelle (mit Paritybit) sind alle nicht erlaubten Bitkombinationen gekennzeichnet.

Das Zeichen ^ (dez. 127) ist in der Code-Tabelle und in der Übersichtstabelle mehrfach vorhanden und entsprechend gekennzeichnet; dem nicht gekennzeichneten ist der Vorzug zu geben.

Die Übersichtstabelle ist eine Darstellung ohne Paritybit. Sie dient nur der Übersicht. Auf sie wird nicht Bezug genommen.

3. Druckersteuerung

Von jeder Zeichenfolge, die dem Drucker übergeben wird, ist das erste Zeichen ein Vorschubsteuerzeichen. Die Bedeutung dieses Steuerzeichens ist im Anhang zu dieser Norm zu entnehmen. Die weiteren Zeichen stellen mit Ausnahme der Zeichen NUL und ZE die Druckinformation einer Zeile dar. Werden mehr Zeichen angeliefert als Schreibstellen in der Zeile vorhanden sind, so werden die überzähligen Zeichen wie das Steuerzeichen NUL behandelt.

Die in der Code-Tabelle angegebenen Steuerzeichen haben folgende Bedeutung:

NUL Nil (Null)

Dieses Zeichen ist ein Füllzeichen. Es wird vom Drucker ignoriert, tritt also im Druckbild nicht in Erscheinung.

SP Zwischenraum (Space)

Dieses Zeichen bewirkt, daß die zugehörige Druckposition unbedruckt bleibt.

ZE Zeilenende

Dieses Zeichen zeigt an, daß für diese Zeile alle Zeichen übergeben sind. Alle folgenden Zeichen werden ignoriert.

4. Zeichenanordnung auf der Druckwalze

Die Zeichen sind in der folgenden Reihenfolge auf dem Umfang der Druckwalze angeordnet:

Reihe Zeichen Code Reihe Zeichen Code Reihe Zeichen Code

1	0	176
2	1	49
3	2	50
4	3	179
5	4	52
6	5	181
7	6	182
8	7	55
9	8	56
10	9	185
11	,	44
12	-	173
13	□	35
14	.	174
15	A	193
16	B	194
17	C	67
18	D	196
19	E	69
20	F	70
21	G	199
22	H	200
23	I	73
24	J	74
25	K	203
26	L	76
27	M	205
28	N	206
29	O	77
30	P	208
31	Q	81
32	R	82
33	S	211
34	T	84
35	U	213
36	V	214
37	W	87
38	X	88
39	Y	217
40	Z	218
41	+	171
42	→	162
43	*	42

44	/	47
45	=	61
46	(168
47)	41
48	:	186
49	;	59
50	'	167
51	[164
52	—	223
53]	64
54	!	161
55	?	191
56	<	188
57	>	62
58	€	38
59	%	37
60	l	94
61	À	91
62	Ö	220
63	Ü	93
64	ß	254
65	0	176
66	1	49
67	2	50
68	3	179
69	4	52
70	5	181
71	6	182
72	7	55
73	8	56
74	9	185
75	,	44
76	-	173
77	.	174
78	E	69
79	a	97
80	b	98
81	c	227
82	d	100
83	e	229
84	f	230
85	g	103
86	h	104

87	i	233
88	j	234
89	k	107
90	l	236
91	m	109
92	n	110
93	o	239
94	p	112
95	q	241
96	r	242
97	s	115
98	t	244
99	u	117
100	v	118
101	w	247
102	x	248
103	y	121
104	z	122
105	ä	251
106	ö	124
107	ü	253
108	à	224
109	#	1
110	"	2
111	{	8
112	}	137
113	~	11
114	^	127
115	§	138
116	§	131
117	v	133
118	^	134
119	~	140
120	<	155
121	>	4
122	^	31
123	π	28
124	—	158
125	r	14
126	^	143
127	°	157
128	10.	7

N3/N

N31/V23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2 - 6



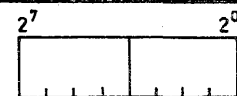
Frei Ausgabem:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

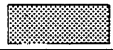
Anmerkungen:

Rm

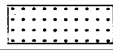
Code-Tabelle (mit Paritybit)



0000	000L	00LO	00LL	0LOO	0LOL	0LLO	0LLL	L000	L00L	L0LO	L0LL	LL00	LL0L	LLLO	LLLL		
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
	16 ^	32 SP	48	64]	80	96	112 p	128 NUL	144	160	176 0	192	208 P	224 @	240	0	0000
1	#	17	33	49 1	65	81 Q	97 a	113	129	145 ^	161 !	177	193 A	209	225	241 q	1 000L
2	"	18	34	50 2	66	82 R	98 b	114	130	146 ^	162 _	178	194 B	210	226	242 r	2 00LO
3	19 ^	35	51	67 C	83	99	115 s	131 \$	147	163	179 3	195	211 S	227	243 c	3 00LL	
4	>	20	36	52 4	68	84 T	100 d	116	132	148 ^	164 [180	196 D	212	228	244 t	4 0LOO
5	21 ^	37	53	69 E	85	101	117 u	133 V	149	165	181 5	197	213 U	229	245 e	5 0LOL	
6	22 ^	38	54	70 F	86	102	118 v	134 ^	150	166	182 6	198	214 V	230	246 f	6 0LLO	
7	10	23	39	55 7	71	87 W	103 g	119	135	151 ^	167 ,	183	199 G	215	231	247 w	7 0LLL
8	{	24	40	56 8	72	88 X	104 h	120	136	152 ^	168 (184	200 H	216	232	248 x	8 L000
9	25 ^	41	57	73 I	89	105	121 y	137 }	153	169	185 9	201	217 Y	233	249 i	9 L00L	
10	26 ^	42	58	74 J	90	106	122 z	138 §	154	170	186 :	202	218 Z	234	250 j	A L0LO	
11	\	27	43	59 ;	75	91 Ä	107 k	123	139	155 <	171 +	187	203 K	219	235	251 ä	B L0LL
12	π	28	44	60	76 L	92	108	124 ö	140 ~	156	172 <	188	204	220	236	252 l	C LLOO
13	ZE	29	45	61 =	77	93 Ü	109 m	125	141	157 °	173 -	189	205 M	221	237	253 ü	D LL0L
14	/	30	46	62 >	78	94 I	110 n	126	142	158 -	174 .	190	206 N	222	238	254 ß	E LLLO
15	/	31	47 /	63	79 O	95	111	127 ^	143 \	159	175	191 ?	207	223	239 o	255	F LLLL



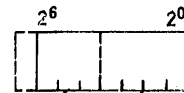
nicht erlaubte Bitkombination



mehrfach im Code enthalten

037 99


Übersichtstabelle



000	00L	0LO	0LL	L00	L0L	LL0	LLL		
0	1	2	3	4	5	6	7		
0 NUL	16 ^	32 SP	48 0	64 J	80 P	96 @	112 p	0	0000
1 #	17 ^	33	49 1	65 A	81 Q	97 a	113 q	1	000L
2 "	18 ^	34 -	50 2	66 B	82 R	98 b	114 r	2	00LO
3 \$	19 ^	35 □	51 3	67 C	83 S	99 c	115 s	3	00LL
4 >	20 ^	36 [52 4	68 D	84 T	100 d	116 t	4	0L00
5 v	21 ^	37 %	53 5	69 E	85 U	101 e	117 u	5	0LOL
6 ^	22 ^	38 &	54 6	70 F	86 V	102 f	118 v	6	0LLO
7	23 ^	39 '	55 7	71 G	87 W	103 g	119 w	7	0LLL
10 {	24 ^	40 (56 8	72 H	88 X	104 h	120 x	8	L000
9 }	25 ^	41)	57 9	73 I	89 Y	105 i	121 y	9	L00L
10 §	26 ^	42 *	58 :	74 J	90 Z	106 j	122 z	A	LOLO
11 \	27 <	43 +	59 ;	75 K	91 Ä	107 k	123 ä	B	LOLL
12 ~	28 π	44 ,	60 <	76 L	92 Ö	108 l	124 ö	C	LL00
13 ZE	29 °	45 -	61 =	77 M	93 Ü	109 m	125 ü	D	LL0L
14 r	30 -	46 .	62 >	78 N	94 I	110 n	126 ß	E	LLL0
15 ʀ	31 /	47 /	63 ?	79 O	95 -	111 o	127 ^	F	LLLL

GR 40d

entspricht dem Code DC1

 mehrfach im Code vorhanden

5. Form der Schriftzeichen



176



49



50



179



52



181



182



55



56



185



193



194



67



196



69



70



199



200



73



74



203



76



205



206



79



208



81



82



211



84



213



214



87



88



217



218



32



171



173



42



47



61



168



41



174



44



167



161



164



64



186



59



38



35



37



188



62



223



162



94



191



91



220



93

GR 155



97



98



227



100



229



230



103



104



233



234



107



236



109



110



239



112



241



242



115



244



117



118



247



248



121



122



251



124



253



26



224



1



2



14



143



134



133



7



8



137



138



28



11



127



31



140



158



157



254



155



4

GR 156

Anhang 1

Für den Anelex-Drucker (SDR 176-1) hat das Vorschubsteuerzeichen (siehe Abschnitt 3) folgenden Aufbau:

Parity-bit	a	0	0	0	b
------------	---	---	---	---	---

Ist das Bit $a = 0$, so wird gemäß dem Vorschublochstreifen vorgeschoben. Der Wert $b + 1$ gibt die Spur-Nummer (1 bis 8) an.

Ist das Bit $a = 1$, so wird das Papier um b Zeilen vorgeschoben.

Bezüglich des Paritybits siehe Abschnitt 2.



Informationsverarbeitung
Druckercode
DC4

2N 0813

.314

Einsprüche bis 31. August 1971

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe AG3: C o d e s, Schriftführer: Böckmann N3/V23, Tel. 2474

1. Zweck und Anwendung

Der Druckercode DC4 bestimmt die Steuerung des Schnell-druckers und gibt die Zuordnung zwischen Bitmuster und auszudruckendem Zeichen an.

Der Zeichenvorrat ist identisch mit der internationalen Version des ISO-7-Bit-Codes gemäß ISO Recommendation R 646 (revised). Siehe dazu auch unter "Erläuterungen".

NUL Nil (Null)

Dieses Zeichen ist ein Füllzeichen. Es wird vom Drucker ignoriert, tritt also im Druckbild nicht in Erscheinung.

2. Code-Tabelle

Die Bitkombinationen, denen in der Code-Tabelle kein Schriftzeichen zugeordnet ist, werden wie das Steuerzeichen SUB behandelt. Die graphische Darstellung des Zeichens SUB ist gemäß Draft ISO Recommendation 2047 das Symbol ∅

Siehe dazu auch unter "Erläuterungen".

SP Zwischenraum (Space)

Dieses Zeichen bewirkt, daß die zugehörige Druckposition unbedruckt bleibt.

3. Druckersteuerung

Die Zeichen werden zeilenweise übergeben. Das erste Zeichen einer Zeile wird als Vorschubsteuerzeichen interpretiert. (Das Format bedarf noch einer späteren Festlegung.)

Werden mehr Zeichen angeliefert, als Schreibstellen in der Zeile vorhanden sind, so werden die überzähligen Zeichen wie das Steuerzeichen NUL behandelt.

Die in der Code-Tabelle angegebenen Steuerzeichen haben folgende Bedeutung:

ZE Zeilenende

Dieses Zeichen zeigt an, daß für diese Zeile alle Zeichen übergeben sind. Alle folgenden Zeichen werden ignoriert.

4. Form der Schriftzeichen

Die Form der Schriftzeichen entspricht der Schrift OCR-B.

N3/N

N31/V23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2

V

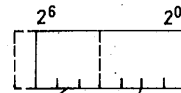
here Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

ferungen:

Rm

Code -Tabelle DC4



000	00L	0LO	0LL	L00	L0L	LL0	LLL		
0	1	2	3	4	5	6	7		
⁰ NUL	¹⁶	³² SP	⁴⁸ 0	⁶⁴ @	⁸⁰ P	⁹⁶ `	¹¹² p	0	0000
¹	¹⁷	³³	⁴⁹ 1	⁶⁵ A	⁸¹ Q	⁹⁷ a	¹¹³ q	1	000L
²	¹⁸	³⁴ "	⁵⁰ 2	⁶⁶ B	⁸² R	⁹⁸ b	¹¹⁴ r	2	00LO
³	¹⁹	³⁵ #	⁵¹ 3	⁶⁷ C	⁸³ S	⁹⁹ c	¹¹⁵ s	3	00LL
⁴	²⁰	³⁶ ⌘	⁵² 4	⁶⁸ D	⁸⁴ T	¹⁰⁰ d	¹¹⁶ t	4	0LO0
⁵	²¹	³⁷ %	⁵³ 5	⁶⁹ E	⁸⁵ U	¹⁰¹ e	¹¹⁷ u	5	0LOL
⁶	²²	³⁸ &	⁵⁴ 6	⁷⁰ F	⁸⁶ V	¹⁰² f	¹¹⁸ v	6	0LLO
⁷	²³	³⁹ /	⁵⁵ 7	⁷¹ G	⁸⁷ W	¹⁰³ g	¹¹⁹ w	7	0LLL
⁸	²⁴	⁴⁰ (⁵⁶ 8	⁷² H	⁸⁸ X	¹⁰⁴ h	¹²⁰ x	8	L000
⁹	²⁵	⁴¹)	⁵⁷ 9	⁷³ I	⁸⁹ Y	¹⁰⁵ i	¹²¹ y	9	L00L
¹⁰	²⁶	⁴² *	⁵⁸ :	⁷⁴ J	⁹⁰ Z	¹⁰⁶ j	¹²² z	A	LOLO
¹¹	²⁷	⁴³ +	⁵⁹ ;	⁷⁵ K	⁹¹ [¹⁰⁷ k	¹²³ }	B	LOLL
¹²	²⁸	⁴⁴ ,	⁶⁰ <	⁷⁶ L	⁹² \	¹⁰⁸ l	¹²⁴	C	LL00
¹³ ZE	²⁹	⁴⁵ -	⁶¹ =	⁷⁷ M	⁹³]	¹⁰⁹ m	¹²⁵ {	D	LL0L
¹⁴	³⁰	⁴⁶ .	⁶² >	⁷⁸ N	⁹⁴ ^	¹¹⁰ n	¹²⁶ -	E	LLL0
¹⁵	³¹	⁴⁷ /	⁶³ ?	⁷⁹ O	⁹⁵ _	¹¹¹ o	¹²⁷ DEL	F	LLLL

GR322

Leere Felder werden wie das Steuerzeichen SUB behandelt und durch das Symbol ⌘ dargestellt.

Erläuterungen

Für diesen Code wurde die Internationale Version des ISO-7-Bit-Codes gemäß ISO Recommendation R 646 (z. Z. in Überarbeitung) zugrunde gelegt. Sie weicht von DIN 66003 (nicht geklammerte Schriftzeichen) und der amerikanischen Norm ASCII nur in einem Schriftzeichen ab: Anstelle des Dollarzeichens wird das internationale Währungszeichen ⌘ (Sun-Symbol) verwendet.

Für das Steuerzeichen SUB ist international das Symbol ⌘ vorgeschlagen. Wenn dieses Zeichen nicht darstellbar ist, muß als Ersatz eines der vorhandenen Schriftzeichen gewählt werden.



Informationsverarbeitung
Tastaturcode
TC1

2N 0813

. 411

Einsprüche bis 20. Dez. 1968

Dieser 1. Normentwurf wird zur Stellungnahme vorgelegt. Der Inhalt kann sich noch in einigen Teilen ändern. Etwaige Einsprüche bzw. Änderungsvorschläge werden erbeten an

Arbeitsgruppe 3 : Codes , Schriftführer: Böckmann N3/GR/V23, Tel. 474

Die Tastatur ist speziell für die Zusammenarbeit mit dem Sichtgerät vorgesehen.

Zeichenvorrat: 87 Druckzeichen
gemäß Zeichensatz ZS 61 und
26 Kleinbuchstaben

								Umschaltung			
								klein ←	→ groß		
000	00L	0LO	0LL	L00	L0L	LLO	LLL	<table border="1"> <tr> <td>2⁶</td> <td>2⁰</td> </tr> </table>		2 ⁶	2 ⁰
2 ⁶	2 ⁰										
0	1	2	3	4	5	6	7				
0	16 g	32 w	48	64 /	80 G	96 W	112 F0	0	0000		
1	17 h	33 x	49	65 ;	81 H	97 X	113 F1	1	000L		
2	18 i	34 y	50	66 DEL	82 I	98 Y	114 F2	2	00LO		
3	19 j	35 z	51	67 =	83 J	99 Z	115 F3	3	00LL		
4	20 k	36 [52	68 %	84 K	100 <	116 F4	4	0L00		
5	21 l	37]	53	69 &	85 L	101 >	117 F5	5	0L0L		
6	22 m	38 ML	54	70 (86 M	102 MO	118 F6	6	0LLO		
7	23 n	39	55	71)	87 N	103 LF	119 F7	7	0LLL		
8	24 o	40	56	72 ,	88 O	104	120 F8	8	L000		
9	25 p	41	57	73	89 P	105	121 F9	9	L00L		
10	a	26 q	42	58	74 A	90 Q	106 SP	122 F10	A	LOLO	
11	b	27 r	43	59	75 B	91 R	107 TAB	123 F11	B	LOLL	
12	c	28 s	44 -	60 -	76 C	92 S	108 +	124 ↵	C	LL00	
13	d	29 t	45 *	61 MR	77 D	93 T	109 :	125 MU	D	LL0L	
14	e	30 u	46 ,	62	78 E	94 U	110 ?	126 ◻	E	LLLO	
15	f	31 v	47 .	63 DEL	79 F	95 V	111 !	127 INS	F	LLLL	

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden.

Steuerzeichen	Taste	Benennung	Steuerzeichen	Taste	Benennung
MR	⇒	Marke nach rechts	TAB	TAB	Tabulator
ML	⇐	Marke nach links	INS	INS	Einsetzen (Insert)
MU	⇓	Marke nach unten	-	REP	Wiederholen (Repeat)
MO	⇑	Marke nach oben	F0	F0	} Funktionstaste 0 bis 11 (Bedeutung nach Vereinbarung)
DEL	DEL	Löschen (Delete)	bis	bis	
LF	↵	Zeilenvorschub mit Wagenrücklauf (Line Feed)	F11	F11	
SP	SP	Zwischenraum (Space)			

N3/NM

GR/V23

Ersatz für:
Ersetzt durch:

Fortsetzung Seite 2

V

Rm
26.11.68

1. Steuerung des Lichtpunktes

Mit den folgenden 4 Tasten kann ein Lichtpunkt über den Schirm des Sichtgerätes geführt werden. An der Stelle, an der der Lichtpunkt steht, kann ein Zeichen gelöscht, doppelt geschrieben, geändert oder eingefügt werden.

Der Lichtpunkt wird bewegt bei

⇒ um ein Zeichen nach rechts

⇐ um ein Zeichen nach links

↓ um eine Zeile nach unten

↑ um eine Zeile nach oben

Die Stelle, an der der Lichtpunkt steht, ist die aktuelle Schreibposition. An dieser Stelle kann ein Zeichen gelöscht, überschrieben oder eingefügt werden.

2. Formatsteuerung

Neben der Steuerung des Formats durch die Bewegung des Lichtpunktes zur Einstellung der Position des nächsten Zeichens stehen noch die folgenden Steuerzeichen zur Verfügung

SP Zwischenraum (Space)

LF Zeilenvorschub (Line Feed) mit Wagenrücklauf (Taste ↵)

TAB Tabulator; Fortbewegung des Lichtpunktes auf die nächste Tabulatorposition, auch über mehrere Zeilen hinweg.

3. Sonstige Steuerzeichen

DEL Löschen (Delete)

Das auf der Position stehende Zeichen wird durch das Zeichen DEL ersetzt. Es ist ein Füllzeichen.

INS Einsetzen (Insert)

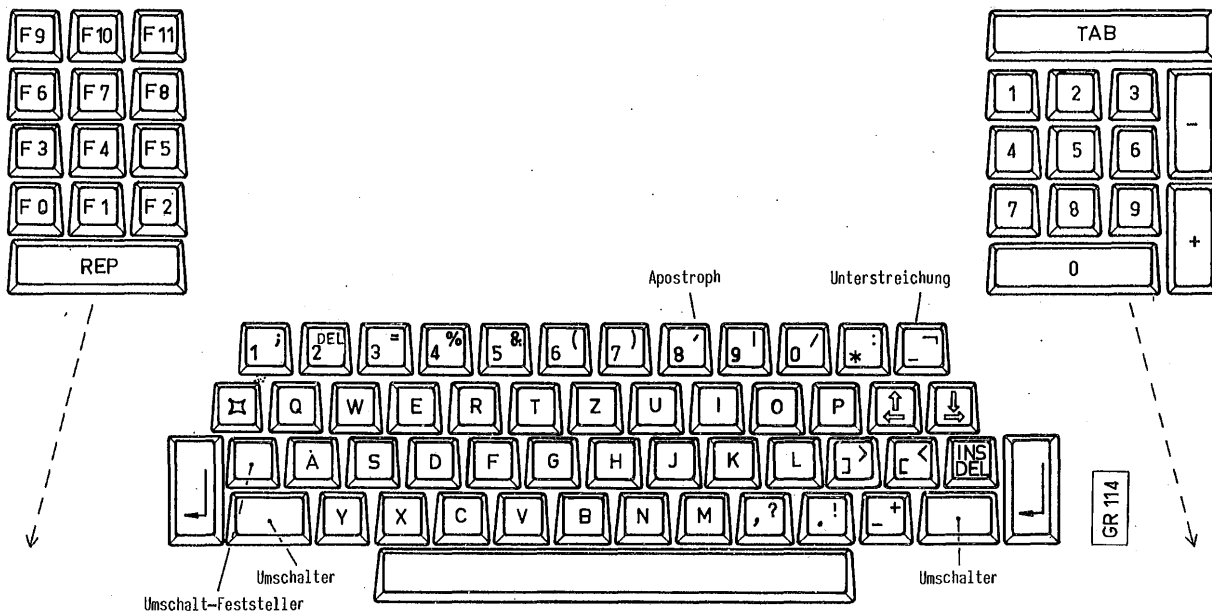
Es wird ein Platz geschaffen, auf den ein Zeichen eingesetzt werden kann. Das Zeichen, das auf dieser Position steht und die folgenden Zeichen werden um eine Position nach rechts versetzt.

4. Wiederholungstaste

Ist die Taste REP gedrückt, so ist jede Taste des Tastenfeldes auf Wiederholung geschaltet, d.h. das Zeichen wird wiederholt so lange eingegeben, wie die Taste gedrückt wird.

5. Funktionstasten

Die Bedeutung der Funktionstasten F0 bis F11 ist für jedes Programm speziell festzulegen.



Sichtgeräte-Tastatur (Die drei Tastenfelder liegen in gleicher Höhe nebeneinander)



Informationsverarbeitung Verschlüsselter Binärcode

2N 0811

VBC 1

.111

1. Zweck und Anwendung

Der verschlüsselte Binärcode (VBC) ermöglicht es, Inhalte von Speichern in einer gesicherten Form auf externen Speichermedien darzustellen.

Die Darstellung auf dem externen Speichermedium ist nicht Gegenstand dieser Norm, sondern ist getrennten Normblättern zu entnehmen.

2. Darstellung

Die Information eines Wortes wird, beginnend mit dem höchstwertigen (linken) Bit in Hexaden aufgeteilt. Ein Wort, dessen Länge nicht ein Vielfaches von 6 Bits ist, wird links durch soviel 0-Bits verlängert gedacht, bis ein Vielfaches von 6 Bits erreicht ist.

Die Darstellung des VBC-Codes erfolgt in Oktaden. Es wird unterschieden zwischen

- Informationsoktaden
- Steueroktaden
- Kopfoktaden

2.1. Informationsoktaden

Die Informationsoktade nimmt die zu speichernde Information auf. Die Informationsoktaden eines Speicherwortes werden durch die Steueroktade Wortende WEx (siehe Abschnitt 3) abgeschlossen. Die Informationsoktade hat folgenden Aufbau:

Hexade der Speicherinformation	0	Paritätsbit
--------------------------------	---	-------------

Die Hexade der Speicherinformation wird rechts durch ein 0-Bit erweitert (Kennzeichen für Informationsoktade) und durch das Paritätsbit so ergänzt, daß die Informationsoktade eine ungerade Anzahl von L-Bits enthält.

Führende (links stehende) Hexaden eines Wortes, die nur 0-Bits enthalten, können entfallen. In diesem Falle wird auch keine Informationsoktade erzeugt. Besteht ein Wort nur aus 0-Bits, so ist lediglich die Steueroktade Wortende WEx erforderlich.

2.2. Steueroktade

Die Steueroktaden haben folgenden Aufbau:

Schlüssel (siehe Tabelle 1)	L	Paritätsbit
-----------------------------	---	-------------

Die Bedeutung der Steueroktade ergibt sich aus dem Schlüssel (siehe Tabelle 1). Die 6 Bits des Schlüssels werden rechts durch ein L-Bit erweitert (Kennzeichen für Steueroktade) und durch das Paritätsbit so ergänzt, daß die Steueroktade eine ungerade Anzahl von L-Bits enthält.

Die Bedeutung der Steueroktade ist in Abschnitt 3 erläutert.

2.3. Kopfoktaden

Die Kopfoktaden enthalten den Kopftext. Er steht immer am Anfang der Gesamtinformation hinter der ersten Steueroktade Z16 (siehe Abschnitt 4). Die Kopfoktade hat den gleichen Aufbau wie die Informationsoktade. Die in ihr enthaltenen Hexaden bilden jedoch nicht einen Teil der Speicherinformation, sondern bilden den Kopftext.

Die Steueroktade WH (Wiederholung) ist im Kopftext nicht erlaubt.

Der Kopftext wird von dem Programm interpretiert, das die im VBC-Code verschlüsselte Information wieder in die ursprüngliche Form zurückverwandelt.

Der Aufbau des Kopftextes ist in Abschnitt 5 erläutert.

3. Bedeutung der Steueroktaden

Hier nicht angegebene Schlüssel dürfen nicht verwendet werden. Wird ein weiterer Schlüssel benötigt, so ist diese Norm zu ergänzen.

WEx Wortende

Die Steueroktaden WE0 bis WE3 zeigen das Ende eines Wortes an.

Zusätzlich können die Steueroktaden eine Wortkennzeichnung angeben. Welche Bedeutung die vier möglichen Kennzeichnungen haben, ist dem Ladeschlüssel des Kopftextes zu entnehmen (siehe Abschnitt 5).

Ist keine Wortkennzeichnung vorgesehen, so sind die vier Steueroktaden gleichwertig.

N3/N

N31/V23

Böckmann

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2 - 3

V

Früher Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

ingen:

Rm

KA Kopftext Anfang

Die folgenden Oktaden sind Kopftoktaden.

KE Kopftext Ende

Die folgenden Oktaden sind Informationsoktaden.

EA Ende der Aufzeichnung

Diese Steueroktade zeigt das Ende der gesamten Information an. Es können nur noch Steueroktaden Fullinformation FU folgen.

FU Fullinformation

Diese Steueroktade wird stets ignoriert. Sie ist erforderlich, um nach der Steueroktade Ende der Aufzeichnung EA einen Block aufzufüllen (siehe auch Abschnitt 4).

WH Wiederholung

Diese Steueroktade schließt ein Wort ab, das einen Wiederholungsfaktor enthält. Es ist eine binär dargestellte positive ganze Zahl, die angibt, wie oft das vorangegangene Wort wiederholt werden soll.

Der Wert 1 bedeutet, daß das Wort insgesamt 2 mal vorkommt. Die vorhergehende Steueroktade darf nicht WH sein.

Zx Gruppennummer (Zähler)

Diese Steueroktade muß stets als erste in einer VBC-Gruppe stehen. Die erste Gruppe erhält die Steueroktade Z16, die folgenden Gruppen entsprechend die nächsthöheren Steueroktaden. Nach Z63 wird wiederum mit Z16 fortgefahren.

Tabelle 1 Schlüssel der Steueroktade

dezimal	Zeichen	Benennung
0	WE0	
1	WE1	Wortende WEx mit Wortkennung 0, 1, 2 oder 3
2	WE2	
3	WE3	
4	KA	Kopftext Anfang
5	KE	Kopftext Ende
6	EA	Ende der Aufzeichnung
7	FU	Fullinformation
8	WH	Wiederholung
9		
10		
11		
12		
13		
14		
15		
16 : : 63	Z16 Z63	Gruppennummer (Zähler) 16 bis 63

4. Aufbau der VBC-Information

Jeweils 108 VBC-Oktaden werden zu einer VBC-Gruppe zusammengefaßt. Die erste Oktade ist die Steueroktade Zx. Sie bildet eine Gruppennummer. Die erste Gruppe erhält die Gruppennummer Z16 und jeweils die folgende Gruppe die nächst höhere Gruppennummer (Z17, Z18, Z19 usw.). Nach der Gruppennummer Z63 wird wiederum mit der Gruppennummer Z16 fortgefahren.

Die Gesamtheit der VBC-Gruppen bildet die VBC-Information. Sie hat folgenden Aufbau:

Steueroktade Z16
Steueroktade KA
Kopftoktaden und
Steueroktaden WEx, Zx
Steueroktade KE
Informationsoktaden und
Steueroktaden WEx, Zx, WH
Steueroktade EA
ggf. Steueroktaden FU

Die Darstellung der VBC-Information auf den Datenträgern ist in getrennten Werknormblättern aufgeführt.

5. Kopftext

Der Kopftext gehört nicht zur Speicherinformation. Er enthält Steuerinformation für das Programm, das die im VBC-Code verschlüsselte Information in die ursprüngliche Form zurückverwandelt.

Der Aufbau des Kopftextes wird durch den im Kopftext enthaltenen Ladeschlüssel bestimmt. Der Ladeschlüssel ist eine binär dargestellte positive ganze Zahl. Durch den Ladeschlüssel wird folgendes festgelegt:

- Länge der Wörter
- Bedeutung der Wortkennung bei den Steueroktaden WEx
- Anzahl der Wörter im Kopftext (der Rest des Kopftextes kann beliebige Information sein).
- Bedeutung der Kopftextwörter

Der Ladeschlüssel ist Bestandteil dieser Norm. Die hier nicht angegebenen Schlüssel dürfen nicht verwendet werden. Wird ein weiterer Schlüssel benötigt, so ist diese Norm zu ergänzen.

Tabelle 2 Ladeschlüssel

Schlüssel	Bedeutung	Kopftext	
0	für Objekte des TR 440 Wartungssystems oder TR 440 - BS 3	siehe Tabelle 3 Version 1	
1			
2			zusätzlich 1024 Ganzwörter - beginnend mit einer Seitengrenze - reservieren
4			
6			
8			
10			
14			
15			Abwickler-Gebiete des BS3

Tabelle 3 Version 1

WEx: x = Typenkennung des Wortes (WE0 = 0, WE1 = 1, WE2 = 2, WE3 = 3)

Wortlänge: 48 Bits

Ursprungscode: ZC1 (für Kopftext)

Erkennungsmaske Länge = 1 Wort, (für Magnetband): Typenkennung = 3, Information (in Tetraden) = FF00FF00FF00

Kopftext:

3	a	a	a	a	a	a	a
3	a	a	a	a	a	a	a
1	0 bzw. 1						
3	u	u	V	.	t	t	t
3	.	m	m	.	i	i	i

} Name

} Ladeschlüssel (binär)

} Datum

u Zwischenraum
a A bis Z und u
t Tag
m Monat
i Jahr

} Ziffern 0 bis 9

Kennblock-Kopftexte: Name = PROGRAMMuuuu
Datum = Datum der Banderstellung

Informations-Kopftext: Name = Name der Information
Datum = Datum der Erstellung des VBC1-Objektes

Schlußblock-Kopftext: Name = ZZZZZZZZZZZZ
Datum = Datum der Banderstellung



Informationsverarbeitung Darstellung des VBC1-Codes auf Lochkarten

2N 0813

.151

1. Zweck und Anwendung

Diese Werknorm legt die Darstellung der Oktaden des VBC1-Codes (Werknorm 2N0811.111) auf Lochkarten fest.

2. Anordnung der Oktaden

Die erste Oktade einer Karte wird in den Zeilen 12 bis 5 der Spalte 1 abgelocht, die zweite in den Zeilen 6 bis 9 der Spalte 1 und den Zeilen 12 bis 1 der Spalte 2, die dritte in den Zeilen 2 bis 9 der Spalte 2, usw. Somit werden für jeweils 3 Oktaden 2 Spalten in der Lochkarte benötigt.

Für die VBC-Information werden die Spalten 1 bis 72 verwendet. Das ergibt 108 Oktaden pro Karte.

Werden die 72 Spalten der letzten Karte nicht alle für die VBC-Information benötigt, so wird bis zur 72. Spalte mit der Steueroktade Füllzeichen FU aufgefüllt.

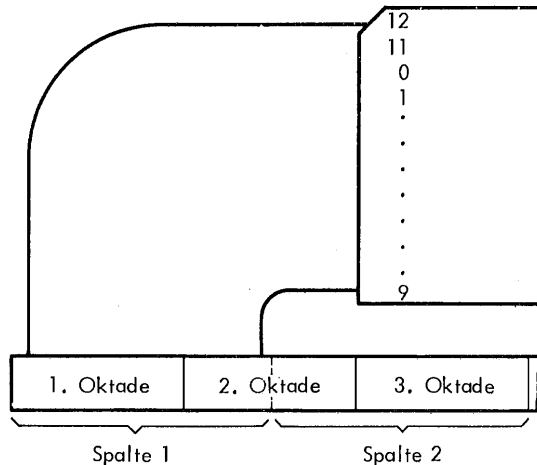
Der Wert L eines Bits in einer Oktade wird auf der Lochkarte durch ein Loch in der zugehörigen Position dargestellt.

Falls die ersten vier Spalten leer (d.h. nicht gelocht) sind, wird die Karte vom einlesenden Programm ignoriert.

3. Numerierung der Karten

Die Spalten 73 bis 76 sind für den Namen, der linksbündig abgelocht wird, vorgesehen. In den Spalten 77 bis 80 wird eine 4-stellige laufende Nummer abgelocht. Sie beginnt auf der ersten Karte mit dem Wert 0000 und wird bei jeder folgenden Karte um den Wert 1 erhöht. Es müssen alle vier Stellen der laufenden Nummer abgelocht werden (auch führende Nullen).

Die Ziffer 1 wird durch ein Loch in der Zeile 1, die Ziffer 2 durch ein Loch in der Zeile 2 usw. dargestellt.



Ausgaben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

ungen:

Rm

N3/N

N31/V23
B. B. B.

Ersatz für:
Ersetzt durch:

V



Informationsverarbeitung
Darstellung des VBC 1-Codes
 auf Lochstreifen

2N 0813

.251

1. Zweck und Anwendung

Diese Werknorm legt die Darstellung der Oktaden des VBC 1-Codes (Werknorm 2N0811.111) auf 8-Spur-Lochstreifen, fest.

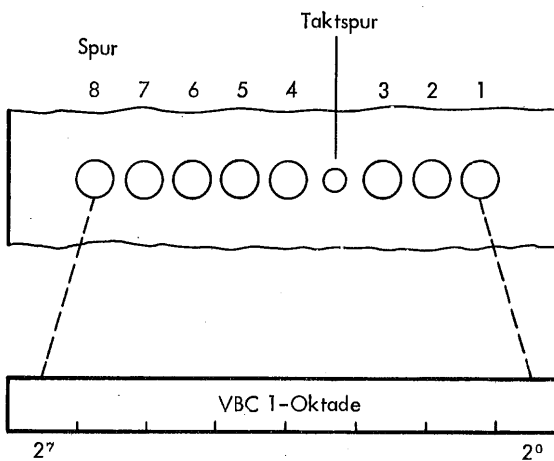
2. Darstellung

Die VBC-Oktaden werden in 8-Spur-Lochstreifen nach DIN 66 016 Blatt 2 abgelocht. Jede Spalte nimmt eine VBC-Oktade auf.

Dem höchstwertigen (linken) Bit einer Oktade wird die Spur 8 zugeordnet und dem niedrigstwertigen Bit die Spur 1.

Der Wert L eines Bits in einer Oktade wird auf der Lochkarte durch ein Loch in der zugehörigen Position dargestellt.

Am Anfang und am Ende jeder Rolle sind mindestens 100 Spalten Leerband (nur Taktloch) zu stanzen.



Ausgeben:

Diese Unterlage darf weder kopiert, noch dritten Personen mitgeteilt, noch anderweitig mißbräuchlich benutzt werden

ungen:

Rm

N3/N

N 31/Y23

Böhm

Ersatz für:

Ersetzt durch:

V



Informationsverarbeitung

Darstellung des VBC1 - Codes

auf 9 - Spur - Magnetband

2N 0813

.651

1. Zweck und Anwendung

Diese Werknorm legt die Darstellung der Oktaden des VBC1-Codes (Werknorm 2N0811.111) auf dem Magnetband fest.

2. Definitionen

Die in dieser Werknorm verwendeten Begriffe "Wort" und "Ursprungscode" sind durch den Ladeschlüssel der Werknorm 2N0811.111, Tabelle 2, definiert.

3. Blockeinteilung

Es müssen folgende Blöcke in der angegebenen Reihenfolge aufgezeichnet werden:

Kennblock
erster Informationsblock
soweit erforderlich weitere Informationsblöcke
mindestens 2 Schlußblöcke

Die Blocklänge beträgt 128 Wörter. Im ersten Wort eines Blockes steht die lfd. Blocknummer und im letzten Wort die gleiche lfd. Blocknummer. Die Blocknummer ist eine positive, ganze Festkommazahl. Die Zählung beginnt beim Kennblock mit der Nummer 0; jeder folgende Block erhält eine um 1 höhere Blocknummer.

3.1. Kennblock

Der Kennblock hat folgenden Aufbau:

1. Wort: Blocknummer 0
2. Wort: Erkennungsmaske
a Wörter: Kopftexte, unverschlüsselt
125-a Wörter: bedeutungslos
128. Wort: Blocknummer 0

Der Kopftext wird in unverschlüsselter Form, d. h. im Ursprungscode aufgezeichnet (siehe Werknorm 2N0811.111). Der Inhalt der auf den Kopftext folgenden Wörter bis auf das letzte Wort, das die Blocknummer enthält, ist bedeutungslos und kann beliebig sein.

3.2. Erster Informationsblock

Der erste Informationsblock hat folgenden Aufbau:

1. Wort: Blocknummer 1
2. Wort: Erkennungsmaske
a Wörter: Kopftexte, unverschlüsselt
17-a Wörter: bedeutungslos
6 x 18 Wörter: VBC1-Gruppen
128. Wort: Blocknummer 1

Erkennungsmaske und Kopftexte sind in der Werknorm 2N0811.111, Tabelle 2, festgelegt. Der Kopftext ist unverschlüsselt, d. h. im Ursprungscode und kann maximal 17 Wörter lang sein. Ist er kürzer, so ist die restliche Information bis zum 19. Wort bedeutungslos. Ab Wort 20 steht die VBC1-Information einschließlich des verschlüsselten Kopftextes. Der erste Informationsblock kann noch 6 VBC1-Gruppen aufnehmen. Sind es weniger, so ist der Rest des Blockes bedeutungslos.

3.3. Weitere Informationsblöcke

Die weiteren Informationsblöcke haben folgenden Aufbau:

1. Wort: Blocknummer n
7 x 18 Wörter: VBC-Gruppen
128. Wort: Blocknummer n

Jeder Block kann außer den Blocknummern noch 7 VBC1-Gruppen aufnehmen. Wird der letzte Block nicht mehr gefüllt, so ist der Rest des Blockes bedeutungslos.

3.4. Schlußblock

Der Schlußblock hat einen ähnlichen Aufbau wie der Kennblock

1. Wort: Blocknummer n
a Wörter: Kopftext, unverschlüsselt
125-a Wörter: bedeutungslos
128. Wort: Blocknummer n

Der Kopftext ist unverschlüsselt, d. h. im Ursprungscode. Es sind mindestens 2 Schlußblöcke erforderlich.

N3/N**N31/V23**

Ersatz für:

Ersetzt durch:

V



Informationsverarbeitung

7-Bit-Code

2N 0812

.411 Blatt 1

Vorbemerkung

Die Festlegungen in dieser Werknorm (Blatt 2) stimmen mit den Festlegungen nach DIN 66 003 überein. Sie werden ergänzt durch Erläuterungen (Blatt 1).

3. DIN 66 003

Der Deutsche Normenausschuß DNA hat den von der ISO empfohlenen 7-Bit-Code unter der Bezeichnung DIN 66 003 genormt.

Die Norm stimmt mit der ISO-Empfehlung überein. Die nationalen Plätze sind gemäß der umstehenden Tabelle belegt.

Verhältnis zu anderen Normen bzw. Empfehlungen

Die Angaben in Blatt 1 dieser Werknorm sollen nur aufzeigen, in welchem Verhältnis DIN 66 003 zu anderen nationalen und internationalen Normen und Empfehlungen über den 7 Bit-Code steht.

4. ECMA

Der Europäische Verband der Hersteller von Rechenmaschinen ECMA (European Computer Manufactures Association) hat den 7-Bit-Code unter der Bezeichnung ECMA-6

ECMA Standard for a
7 Bit Input/Output Character Code
June 1967 - 2nd Edition

veröffentlicht. Er entspricht dem ISO-7-Bit-Code mit reservierten Plätzen für nationale Zeichen.

1. ISO

Die ISO (International Organization for Standardization), die Dachorganisation der nationalen Normengremien, hat den 7-Bit-Code unter der Bezeichnung

ISO Recommendation R 646
6 and 7 Bit Coded Character Sets for
Information Processing Interchange
1st Edition December 1967

veröffentlicht. In dieser Veröffentlichung sind einige Plätze in der Code-Tabelle für nationale Zwecke reserviert, die von den nationalen Normengremien den Erfordernissen entsprechend belegt werden können.

Es wird ausdrücklich betont, daß für jedes Land die eigene, nationale Norm die einzig verbindliche ist.

5. CCITT 5

Der Internationale beratende Ausschuß für den Telegrafen- und Fernsprehdienst CCITT (Comité Consultatif International Télégraphique et Téléphonique) hat den 7-Bit-Code unter der Bezeichnung

CCITT Recommendation V3
International Alphabet Nr. 5
for transmission of data and messages
October 1968

veröffentlicht. Er entspricht dem ISO-7-Bit-Code mit reservierten Plätzen für nationale Zeichen.

2. ASCII und USA SCI I

Die nationale Normenbehörde USASI (USA Standards Institut) der USA (früher ASA) hat den von der ISO empfohlenen 7-Bit-Code unter der Bezeichnung USA SCI I (früher ASCII) genormt.

Die Norm stimmt mit der ISO-Empfehlung überein. Die nationalen Plätze sind gemäß der umstehenden Tabelle belegt.

6. Andere nationale Normen

Umstehende Tabelle zeigt, wie andere nationale Normengremien die nationalen Plätze belegt haben (eine Aufstellung der ECMA vom Dezember 1968).

N3/N

N 31/V 23

Ersatz für:

Ersetzt durch:

Fortsetzung Seite 2

V

Frühere Ausgaben:

Diese Unterlage darf weder kopiert, noch
dritten Personen mitgeteilt, noch ander-
weitig mißbräuchlich benutzt werden

rungen:

7. 8-Bit-Code

Bei den verschiedenen Normengremien wird an einem 8-Bit-Code gearbeitet. Die Arbeiten stehen noch am Anfang, und bis zur Normung wird noch einige Zeit vergehen.

Es ist jedoch schon jetzt zu erkennen, daß der 8-Bit-Code in den Spalten 0 bis 7 den 7-Bit-Code als Untermenge enthalten wird (8. Bit = 0). Die Spalten 8 und 9 werden wahrscheinlich zusätzliche Steuerzeichen und die Spalten 10 bis 15 zusätzliche Schriftzeichen enthalten.

Spalte		0	7 8 9	10	15
Zeile		7-Bit-Code			zusätzliche Schriftzeichen
0					
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					
13					
14					
15					

Tabelle

Belegung der nationalen Plätze durch die nationalen Normengremien
Aufstellung der ECMA, Stand Dezember 1968

	2/3	4/0	5/11	5/12	5/13	5/14	6/0	7/11	7/12	7/13	7/14
Germany	#	@	[\]	^	~	{		}	—
	f	s	A	ö	ü			ä	ö	ü	ß
Australia											
Belgium	#	@	[\]	^	~	{		}	~
Danemark			A	ö	ä			ä	ö	ä	
U S A	#	@	[\]	^	~	{		}	~
France	f	à	°	ç	ç	^	~	é	ù	è	~
			[\]						
Italy		@	[\]	^	~	{		}	~
		s	l	o	ò	^	à	é	ù	è	~
Japan	#	@	[¥]	^	~	{		}	—
Norway Finland			A	ö	ä			ä	ö	ä	
The Nether- lands		@	[\]	^	~	{		}	~
Polen											
United Kingdom	f	@	[\]	^	~	{		}	—
				10							
				1/2							
Sweden			A	ö	ä			ä	ö	ä	



Informationsverarbeitung 7-Bit-Code

DIN
66 003

Information processing; 7 bit coded character set

2N 0812.411 Bl.2

Zusammenhang mit der ISO-Empfehlung ISO/R 646 - 1967 siehe Erläuterungen.

1. Zweck und Anwendung

Der in dieser Norm festgelegte 7-Bit-Code dient zur Übergabe von digitalen Daten¹⁾ zwischen verschiedenen Datenverarbeitungsanlagen¹⁾ sowie zur Ein- und Ausgabe bei solchen Anlagen. Der Code¹⁾ ordnet Zeichen¹⁾, die aus 7 Bits¹⁾ bestehen, bestimmten Schrift- und Steuerzeichen zu.

Mit dieser Norm ist nicht beabsichtigt, einen maschinen-internen Code festzulegen.

Die Darstellung des Code auf Datenträgern²⁾, wie Lochstreifen und Magnetband, wird in DIN 66 004 festgelegt. Dort wird auch die Sicherung der dargestellten Daten mit Hilfe von Prüfbits und Prüfzeichen in Abhängigkeit vom jeweiligen Datenträger behandelt.

2. Code-Tabelle

2.1. Zeichenvorrat und Codierung

Der 7-Bit-Code ist durch die Code-Tabelle festgelegt.

Der Zeichenvorrat besteht aus Schriftzeichen und Steuerzeichen. Schriftzeichen sind Ziffern, Buchstaben und Sonderzeichen. Zu den Sonderzeichen siehe Abschnitt 2.4.

2.2. Bezeichnung der Codierung

Die einzelnen Zeichen des 7-Bit-Code können durch ihre Bitkombination in der Reihenfolge $b_7, b_6, b_5, b_4, b_3, b_2, b_1$ oder durch die Angabe von Spalten- und Zeilennummer ihres Platzes in der Code-Tabelle gekennzeichnet werden. Beispiel:

Buchstabe F entspricht Bitkombination 1000110
oder Platz 4/6.

Darüber hinaus kann jedem Zeichen des 7-Bit-Code eine Wertigkeit nach dem Dualsystem¹⁾ zugeteilt werden, wobei $b_7 = 2^6, b_6 = 2^5, b_5 = 2^4$ usw. bis $b_1 = 2^0$ zu setzen ist.

2.3. Erklärung der in der Code-Tabelle benutzten internationalen Kurzzeichen

(siehe auch Abschnitt 2.6 und die Erläuterungen am Schluß der Norm; die in Klammern in kursiver Schrift hinzugefügten englischen Benennungen sind der ISO-Empfehlung ISO/R 646 - 1967 entnommen):

- 1) Begriff siehe DIN 44 300 „Informationsverarbeitung, Begriffe“
- 2) Begriff siehe DIN 44 300 Blatt 10 „Informationsverarbeitung, Begriffe, Ergänzung zu DIN 44 300“ (z. Z. noch Entwurf)

Platz (Spalte/Zeile)	Kurzzeichen	Benennung
0/0	NUL	Nil (Null)
0/1 und weitere	TC	Übertragungssteuerung (Transmission Control)
0/1	SOH	Anfang des Kopfes (Start of Heading)
0/2	STX	Anfang des Textes (Start of Text)
0/3	ETX	Ende des Textes (End of Text)
0/4	EOT	Ende der Übertragung (End of Transmission)
0/5	ENQ	Stationsaufforderung (Enquiry)
0/6	ACK	Positive Rückmeldung (Acknowledge)
0/7	BEL	Klingel (Bell)
0/8 bis 0/13	FE	Formatsteuerung (Format Effector)
0/8	BS	Rückwärtsschritt (Backspace)
0/9	HT	Horizontal-Tabulator (Horizontal Tabulation)
0/10	LF	Zeilenvorschub (Line Feed)
0/11	VT	Vertikal-Tabulator (Vertical Tabulation)
0/12	FF	Formularvorschub (Form Feed)
0/13	CR	Wagenrücklauf (Carriage Return)
0/14	SO	Dauerumschaltung (Shift-out)
0/15	SI	Rückschaltung (Shift-in)
1/0	DLE	Datenübertragungs- umschaltung (Data Link Escape)
1/1 bis 1/4	DC	Gerätesteuerung (Device Control)
1/5	NAK	Negative Rückmeldung (Negative Acknowledge)

Fortsetzung Seite 2 bis 5
Erläuterungen Seite 5

Fachnormenausschuß Informationsverarbeitung (FNI) im Deutschen Normenausschuß (DNA)

Platz (Spalte/Zeile)	Kurzzeichen	Benennung
1/6	SYN	Synchronisierung (Synchronous Idle)
1/7	ETB	Ende des Datenübertragungsblocks (End of Transmission Block)
1/8	CAN	Ungültig (Cancel)
1/9	EM	Ende der Aufzeichnung (End of Medium)
1/10	SUB	Substitution (Substitute Character)
1/11	ESC	Umschaltung (Escape)
1/12 bis 1/15	IS	Informationstrennung (Information Separator)
1/12	FS	Hauptgruppen-Trennung (File Separator)
1/13	GS	Gruppen-Trennung (Group Separator)
1/14	RS	Untergruppen-Trennung (Record Separator)
1/15	US	Teilgruppen-Trennung (Unit Separator)
2/0	SP	Zwischenraum (Space)
7/15	DEL	Löschen (Delete)

2.4. Benennung der in der Code-Tabelle vorkommenden Sonderzeichen

Platz (Spalte/Zeile)	Schriftzeichen	Benennung
2/0		Zwischenraum ³⁾
2/1	!	Ausrufungszeichen
2/2	"	Anführungszeichen
2/3	#	Nummernzeichen
2/3	£	Pfund (Währung)
2/4	\$	Dollar
2/5	%	Prozent
2/6	&	kommerzielles Und
2/7	'	Apostroph
2/8	(runde Klammer auf
2/9)	runde Klammer zu
2/10	*	Stern
2/11	+	plus
2/12	,	Komma
2/13	-	minus
2/14	.	Punkt
2/15	/	Schrägstrich
3/10	:	Doppelpunkt
3/11	;	Semikolon
3/12	<	kleiner als
3/13	=	gleich
3/14	>	größer als

³⁾ Dieses Zeichen ist auch ein Formatsteuerzeichen, siehe Abschnitt 2.6.3

⁴⁾ Begriff siehe DIN 44 302 „Datenübertragung, Begriffe“ (z. Z. noch Entwurf)

Platz (Spalte/Zeile)	Schriftzeichen	Benennung
3/15	?	Fragezeichen
4/0	@	kommerzielles à
4/0	§	Paragraph
5/11	[eckige Klammer auf
5/12	\	inverser Schrägstrich
5/13]	eckige Klammer zu
5/14	^	Zirkumflex
5/15	_	Unterstreichung
6/0	`	Gravis
7/11	{	geschweifte Klammer auf
7/12		senkrechter Strich
7/13	}	geschweifte Klammer zu
7/14	~	Überstreichung

2.5. Bedeutung und Form der Schriftzeichen

Die Bedeutung der Schriftzeichen wird durch diese Norm nicht festgelegt. Sie muß je nach Anwendung des 7-Bit-Code vereinbart werden, sofern in einschlägigen Normen, z. B. für Programmiersprachen, nichts darüber ausgesagt ist. Zu vermeiden ist jedoch jede Interpretation, die im Gegensatz zu einer gebräuchlichen Bedeutung steht. Die Benutzung eines Schriftzeichens in mehreren Bedeutungen ist zulässig, z. B. kann das Schriftzeichen minus auch Bindestrich oder Gedankenstrich oder Trennungsstrich bedeuten.

Die Form der Schriftzeichen ist durch diese Norm nicht festgelegt. Zum Beispiel kann der senkrechte Strich (Platz 7/12) auch unterbrochen dargestellt werden.

2.6. Erklärung zu den Schrift- und Steuerzeichen

2.6.1. Schriftzeichen

Den Bitkombinationen, deren Plätze in der Code-Tabelle mit * versehen sind (Plätze 2/3, 4/0, 5/11 bis 5/14, 6/0 und 7/11 bis 7/14), können in anderen Ländern andere Schriftzeichen zugeordnet sein. Deshalb müssen bei internationaler Datenübertragung Vereinbarungen für das Benutzen dieser Bitkombinationen getroffen werden.

Unter Beachtung dieser Einschränkung dürfen nach besonderer Vereinbarung sowohl national als auch international die Bitkombinationen der Plätze 2/3, 4/0, 5/11 bis 5/13 und 7/11 bis 7/14 auch alternativ mit der in Klammern angegebenen Bedeutung benutzt werden.

Das Anführungszeichen (Platz 2/2) kann auch als Trema (für eine Diäresis) und das Zeichen Apostroph (Platz 2/7) auch als Betonungszeichen Akut verwendet werden. Wenn ein Buchstabe mit einem diakritischen Zeichen, wie z. B. Akut, Gravis, Trema, versehen werden soll, dann ist vorzugsweise folgende Reihenfolge einzuhalten: Buchstabe, Rückwärtsstrich (Platz 0/8), diakritisches Zeichen.

2.6.2. Übertragungssteuerzeichen

Diese Zeichen sind ausschließlich dazu bestimmt, den Betriebsablauf der Übertragung von digitalen Daten zwischen Datenstationen⁴⁾ zu steuern.

Anfang des Kopfes (SOH)

Das Zeichen wird am Anfang einer Zeichenfolge verwendet, die eine Adresse und/oder Angaben zur Weiterleitung enthält und maschinell interpretierbar ist.

Anfang des Textes (STX)

Das Zeichen leitet eine zusammengehörige Zeichenfolge ein, die vollständig an den Empfänger übermittelt wird. Eine solche Zeichenfolge wird als „Text“ bezeichnet. Das Zeichen STX beendet eine Zeichenfolge, die mit SOH beginnt.

Ende des Textes (ETX)

Das Zeichen beendet eine Zeichenfolge, die mit STX beginnt.

Ende der Übertragung (EOT)

Das Zeichen beendet die Übertragung von einem oder mehreren Texten.

Stationsaufforderung (ENQ)

Das Zeichen dient zur Anforderung einer Antwort von einer fernen Station. Die Antwort kann die Stationskennung und/oder den Stationszustand einschließen.

Wenn in öffentlichen Netzen eine „Wer da?“-Funktion benötigt wird, dann muß das Zeichen ENQ bei seinem ersten Auftreten, nachdem die Verbindung hergestellt ist, „Wer da?“ (Aufforderung zur Abgabe der Stationskennung) beinhalten. Je nach Vereinbarung können später auftretende Zeichen ENQ diese Funktion beinhalten oder nicht.

Positive Rückmeldung (ACK)

Das Zeichen ist eine bejahende Antwort der Empfangsstation an die Sendestation.

Datenübertragungsumschaltung (DLE)

Das Zeichen leitet eine Zeichenfolge ein, die einen außerhalb des Code liegenden Übertragungssteuerbefehl darstellt. In einer solchen Zeichenfolge dürfen nur Bitkombinationen vorkommen, die in der Code-Tabelle mit Schriftzeichen oder Übertragungssteuerzeichen belegt sind. Die Zeichenfolge und ihre Interpretation bedürfen der Vereinbarung.

Negative Rückmeldung (NAK)

Das Zeichen ist eine verneinende Antwort der Empfangsstation an die Sendestation.

Synchronisierung (SYN)

Das Zeichen wird in Synchron-Systemen übertragen, wenn kein anderes Zeichen zum Senden vorliegt, und dient zum Herstellen oder Erhalten des Synchronismus zwischen den Endstellen einer Verbindung.

Ende des Datenübertragungsblocks (ETB)

Das Zeichen schließt einen Datenübertragungsblock⁴⁾ ab.

2.6.3. Formatsteuerzeichen

Diese Zeichen bestimmen die Anordnung der Daten auf Datenträgern für die Ein- und Ausgabe und damit gegebenenfalls auch ihre Gliederung. Die nachstehend angegebene Bedeutung der einzelnen Zeichen gilt für zeichenweise druckende Geräte (z. B. Schreibmaschinen und Fernschreibergeräte). Bei anderen Geräten dürfen diese Zeichen mit verwandter Bedeutung benutzt werden; hierzu bedarf es aber besonderer Vereinbarungen.

Zwischenraum (SP)

Das Zeichen dient zur Vorwärtsbewegung der Schreibeinrichtung um einen Schreibrschritt innerhalb einer Zeile. Es wird Vorwärtsschritt genannt, wenn es nur auf die Steuerfunktion ankommt.

Rückwärtsschritt (BS)

Das Zeichen veranlaßt die Rückbewegung der Schreibeinrichtung um einen Schreibrschritt innerhalb der Zeile.

Horizontal-Tabulator (HT)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung bis zur nächsten Tabulatorposition innerhalb der Zeile.

Zeilenvorschub (LF)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung um eine Zeile. Bei Geräten, deren Zeilenvorschub mit Wagenrücklauf kombiniert ist, leitet dieses Zeichen die Fortbewegung der Schreibeinrichtung bis zur ersten Schreibposition der nächsten Zeile ein. (Bei Geräten in öffentlichen Wählnetzen ist die Zusammenfassung der beiden Zeichen LF und CR zu einem einzigen Steuerzeichen nicht erlaubt.)

⁴⁾ siehe Seite 3

Vertikal-Tabulator (VT)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung senkrecht zur Zeile.

Formularvorschub (FF)

Das Zeichen dient zur Fortbewegung der Schreibeinrichtung auf die erste Zeile des nächsten Formulars.

Wagenrücklauf (CR)

Das Zeichen dient zur Rückbewegung der Schreibeinrichtung auf die erste Schreibposition innerhalb der Zeile. Bei Geräten, deren Zeilenvorschub mit Wagenrücklauf kombiniert ist, entfällt das Steuerzeichen Wagenrücklauf.

2.6.4. Gerätesteuerzeichen

Diese Zeichen dienen zum Steuern von Zusatz- und Hilfsgeräten in Datenverarbeitungsanlagen und Übertragungssystemen, insbesondere zum Ein- und Ausschalten dieser Geräte. Beispielsweise können sie bei Lochstreifenbetrieb folgendermaßen benutzt werden:

DC 1 – Erster Locher ein

DC 2 – Zweiter Locher ein

DC 3 – Abtaster ein

DC 4 wird zum Abschalten der Geräte verwendet.

Das Benutzen der Gerätesteuerzeichen bedarf besonderer Vereinbarungen.

2.6.5. Informationstrennzeichen

Die vier Trennzeichen

Teilgruppen-Trennung (US)

Untergruppen-Trennung (RS)

Gruppen-Trennung (GS) und

Hauptgruppen-Trennung (FS)

dienen zur logischen Gliederung von Daten. Sie werden in der vorstehenden Rangfolge verwendet, wobei die Teilgruppe die kleinste Einheit ist.

2.6.6. Steuerzeichen zur Code-Erweiterung**Umschaltung (ESC)**

Das Zeichen verändert die Bedeutung des nächstfolgenden Zeichens gegenüber der in der Code-Tabelle angegebenen. Falls erforderlich, kann dieses nächstfolgende Zeichen wiederum eine Zeichenfolge einleiten, die als außerhalb des Code liegend zu interpretieren ist. Die durch Umschaltung eingeleitete Zeichenfolge (das einzelne Zeichen und die gegebenenfalls davon abhängige Zeichenfolge) soll die Zeichen Nil, Löschen und die zehn Übertragungssteuerzeichen nicht enthalten, es sei denn in ihrer ursprünglichen Bedeutung.

Eine durch Umschaltung eingeleitete Zeichenfolge stellt einen außerhalb des Code liegenden Steuerbefehl dar, jedoch keinen Übertragungssteuerbefehl. Solche Steuerbefehle können unter anderem dazu dienen, zusätzliche Schriftzeichen und Alphabete zu gewinnen. Das auf Umschaltung folgende Zeichen, die gegebenenfalls davon abhängige Zeichenfolge und beider Interpretation bedürfen der Vereinbarung.

Dauerumschaltung (SO)

Die auf das Zeichen Dauerumschaltung folgenden Zeichen sind nicht nach der Code-Tabelle zu interpretieren – mit Ausnahme der Steuerzeichen (Spalten 0 und 1 und Platz 7/15). Eine durch Dauerumschaltung und Rückschaltung begrenzte Zeichenfolge dient ausschließlich dazu, zusätzliche Schriftzeichen zu gewinnen. Die Zeichenfolge und ihre Interpretation bedürfen der Vereinbarung.

Rückschaltung (SI)

Die auf das Zeichen Rückschaltung folgenden Zeichen werden nach der Code-Tabelle interpretiert.

2.6.7. Sonstige Steuerzeichen**Nil (NUL)**

Das Zeichen ist ein Füllzeichen. Es kann also einer Zeichenfolge hinzugefügt oder in dieser unterdrückt werden, ohne daß sich ihre Bedeutung ändert; beeinflusst wird nur die Darstellung der Informationen und/oder die Gerätesteuerung.

Klingel (BEL)

Das Zeichen bewirkt ein vorzugsweise akustisches Signal und kann auch Anzeigeräte auslösen.

Ungültig (CAN)

Das Zeichen bedeutet, daß die vorangehenden Zeichen Fehler enthalten oder eliminiert werden sollen. Es ist besonders zu vereinbaren, welcher Abschnitt zu berichtigen ist. Er soll vorzugsweise durch ein Steuerzeichen (z. B. ein Informationstrennzeichen) begrenzt sein.

Ende der Aufzeichnung (EM)

Das Zeichen kann benutzt werden, um das Ende des Datenträgers oder das Ende des gewünschten oder verwendeten Teils der Aufzeichnung auf einem Datenträger anzugeben.

Substitution (SUB)

Das Zeichen dient dazu, ein Zeichen zu ersetzen, das als nicht zulässig oder fehlerhaft erkannt worden ist.

Löschen (DEL)

Das Zeichen wird vorzugsweise zum Auslösen oder Überschreiben fehlerhafter oder nicht gewünschter Zeichen in Lochstreifen benutzt. Es kann auch als Füllzeichen dienen, also einer Zeichenfolge hinzugefügt oder in dieser unterdrückt werden, ohne daß sich ihre Bedeutung ändert; beeinflusst wird nur die Darstellung der Informationen und/oder die Gerätesteuerung.

Erläuterungen

In der ISO ⁵⁾ hat das Technische Komitee TC 97 „Computers and Information Processing“ unter deutscher Beteiligung die Empfehlung ISO/R 646-1967 „6 and 7 Bit Coded Character Sets for Information Processing Interchange / Jeux de caractères codés à 6 et 7 éléments pour l'échange d'information entre matériels de traitement de l'information / 6- und 7-Bit-Code für den Informationsaustausch in der Informationsverarbeitung“ fertiggestellt. Das Ziel dieses Komitees und ebenso das der nationalen Normenausschüsse war es, einen einheitlichen Code zu schaffen, der sowohl den Erfordernissen der Datenverarbeitung als auch der Datenübertragung gerecht wird.

Der in der ISO-Empfehlung enthaltene 7-Bit-Code ist der vorliegenden Norm zugrunde gelegt. Er bietet die Möglichkeit, sinnvoll binär zu sortieren, und enthält die für die verschiedenen Interessentengruppen notwendigen Steuerzeichen. Diese haben niedrigere binäre Wertigkeiten als die Schriftzeichen und sind dadurch von ihnen leicht zu unterscheiden. Nur das Zeichen Löschen hat wegen seines Gebrauchs in der Lochstreifentechnik die Stelle mit der höchsten Wertigkeit.

Der Code enthält das lateinische Alphabet mit Groß- und Kleinbuchstaben. Die Umlaute und der Buchstabe ß, die nur nach besonderer Vereinbarung verwendet werden dürfen, sind jeweils an das Ende der Groß- und Kleinbuchstaben auf die Plätze 5/11 bis 5/13 bzw. 7/11 bis 7/14 gelegt worden, die von der ISO für eine nationale Verwendung vorgesehen sind. Im deutschen Alphabet werden diese Buchstaben beim Sortieren je nach Anwendung den entsprechenden Buchstaben (a, o, u) oder Buchstabenpaaren (æ, oe, ue, ss) gleichgesetzt.

Für das Zeichen Zwischenraum (SP) ist in dieser Norm kein abdruckbares Zeichen angegeben. Im Einzelfall kann es zweckmäßig sein, dieses Zeichen z. B. durch $_$ oder b (aus englisch „blank“) bildlich darzustellen.

Die Ausführungen im Abschnitt 2.5, wonach die Bedeutung der Schriftzeichen nicht festgelegt ist, verdienen besondere Aufmerksamkeit. So ist dadurch zum Beispiel erlaubt, daß in bestimmten Anwendungen Buchstaben als negative Ziffern (bei Lochkarten) oder als Steuerbefehle (für numerische Steuerungen) interpretiert werden.

Die in der Code-Tabelle mit * versehenen Bitkombinationen sind die sogenannten „nationalen“ Plätze (ausgenommen

Platz 2/3). Hier können die verschiedenen Länder ihrer Sprache und ihren besonderen Bedürfnissen entsprechend zusätzliche Buchstaben, Akzente und Sonderzeichen einsetzen. Deshalb ist es notwendig, daß bei internationalem Datenaustausch fallweise für diese Plätze Vereinbarungen getroffen werden.

In der Code-Tabelle der vorliegenden Norm sind den nationalen Plätzen 4/0, 5/11, 5/13, 5/14, 6/0 und 7/14 die Sonderzeichen zugeordnet, die in der ISO-Empfehlung zur vorzugsweisen Anwendung empfohlen werden. Da für die Plätze 5/12 und 7/11 bis 7/13 solche Empfehlungen fehlen, erscheinen hier die Sonderzeichen, die in der amerikanischen Norm als nationale Belegung dieser Plätze vorgesehen sind.

Wie bereits erwähnt, hat in DIN 66 003 eine Reihe der nationalen Plätze deutsche Sonderbuchstaben als zweite Belegung. Ebenfalls vorerst nur in Deutschland vorgesehen ist die zweite Belegung des Platzes 4/0. Das hier alternativ zugelassene Sonderzeichen § ist in den Tastenfeldern nach DIN 2112 und DIN 2127 enthalten.

Auf dem Platz 2/3 ist in der ISO-Empfehlung das Währungszeichen £ angegeben. In Ländern, in denen dieses Währungszeichen nicht benötigt wird, darf statt dessen das Zeichen für Nummer benutzt werden. Dieses ist in der vorliegenden Norm als Vorzugsbelegung gewählt worden, jedoch ist nach Vereinbarung auch das Währungszeichen £ zulässig.

Durch die Benennung „Nil“ (lateinisch: „nichts“) für das Zeichen auf Platz 0/0 (statt „Leer“) werden Verwechslungen mit der „Leertaste“ vermieden, mit der auf Büromaschinen der Zwischenraum (und nicht das Zeichen Nil) erzeugt wird. Zur Bedeutung siehe Abschnitt 2.6.7.

Aus dem Titel der oben zitierten ISO-Empfehlung geht hervor, daß sie auch einen 6-Bit-Code enthält. Dieser wird nicht zur Deutschen Norm vorgeschlagen, weil derzeit offenbar kein Bedarf für einen genormten 6-Bit-Code besteht. Auch das CCITT ⁶⁾ hat sich dafür ausgesprochen, keinen 6-Bit-Code festzulegen. Hingegen will es den 7-Bit-Code der ISO als Internationales Telegraphen-Alphabet Nr 5 übernehmen.

⁵⁾ ISO = International Organization for Standardization

⁶⁾ CCITT = Comité Consultatif International Télégraphique et Téléphonique