

PI

# DATA ENTRY Release 2

## Format Language Guide



36000

**DATA ENTRY**  
**Release 2**  
**Format Language Guide**

A/S REGNECENTRALEN  
Marketing Department

Second Edition  
August 1977  
RCSL 42-i 0664

**DATA ENTRY**  
**Release 2**  
**Format Language Guide**

A/S REGNECENTRALEN  
Marketing Department

Second Edition  
August 1977  
RCSL 42-i 0664

Authors: Aino Andersen, Lis Clement, Peter Jørgensen, Bodil Larsen  
Text Editor: Peter Jørgensen

KEY WORDS: MUS, Data Entry System, batch translation, format language, definitions, coding sheets, execution, examples.

ABSTRACT: This manual contains a description of the format language, and instructions on the writing, translation and execution of format programs.

Users of this manual are cautioned that the specifications contained herein are subject to change by RC at any time without prior notice. RC is not responsible for typographical or arithmetic errors which may appear in this manual and shall not be responsible for any damages caused by reliance on any of the materials presented.

Copyright © A/S Regnecentralen, 1977

Printed by A/S Regnecentralen, Copenhagen

## Foreword

The main differences between the present second edition and the first edition of RC 3600 Data Entry Release 2 Format Language Guide (September 1976 ) are due to the extension of the RC 3600 Data Entry System with

- multiple - entried tables (disc tables)
- subscript - intervals and conversion operators
- pseudo - registers
- infinite number of formats, subprograms and tables.

In order to handle multiple - entried tables the standard format TABLE has been changed and a new coding sheet (disc table coding sheet) has been introduced. It is now possible to give between 0 and 6 functions to one argument, so the syntax of the SEARCH - statement has been changed (variable number of destinations, and one or more destination(s) may be dummy).

To permit reference to groups of characters within fields or registers subscript - intervals and conversion operators (ALPHANUMERIC, NUMERIC) have been introduced.

The system has been extended with a number of pseudo-registers containing run - time information, such as batchname, data and time.

Some instructions concerning (user defined) registers have been made in order to leave fill characters and justifications untouched.

- a. automatic field (i.e.kind D, C or I) must have exactly the same length as the corresponding register.
- b. after execution of an eventual field program connected to a D-field or an I-field the field contents will be copied into the register.
- c. I-fields must be of numeric type (N).

Beside this (user-defined) registers with a negativ value will be in embedded trailing representation (as for SS-fields).

In section 3.7.1.7 DUP-statement is described even if it is not implemented yet. The statement is used to simulate activation of the DUP control key.

Finally the TRANSLATE program has been changed in order to inform about the size of a translated item (i.e. format, image, subprogram and table).

# How to Use this Guide

The purpose of this manual is to enable the programmer to acquire a reliable working knowledge of the Format Language. It is believed that a careful study of the six chapters and the appendices will equip him with all the insight required to write and successfully operate Data Entry format programs.

The subjects dealt with in the various chapters and sections are listed in the Table of Contents. For the novice, especially, the following order of study is considered preferable:

- 0 Introduction
  - 3 The Format Language
  - 1 Definitions
  - 2 Format - Image - Subprogram - Table Coding Sheets
  - 4 Execution of Format Programs
  - 5 Entering New Formats, Subprograms, and Tables
  - 6 Programming Hints
- Appendices I - VII

Appendix VIII Definitions of Terms and IX Index should be consulted any time the user of this manual feels the need to orientate himself about the terminology.





# Table of Contents

0	INTRODUCTION	page 0 - 1
1	DEFINITIONS	1 - 1
1.1	Definition of Batch - Record - Field	1 - 1
1.1.1	Batch	1 - 1
1.1.2	Record	1 - 1
1.1.3	Field	1 - 1
1.2	Definition of Format, Subformat, and Field Description	1 - 3
1.2.1	Format	1 - 3
1.2.2	Subformat	1 - 3
1.2.3	Field Description	1 - 3
1.2.4	Field Definition - Field Program	1 - 3
1.3	Definition of Subprogram - Table	1 - 5
1.3.1	Subprogram	1 - 5
1.3.2	Table	1 - 5
1.3.3	Argument - Function	1 - 5
1.4	Definition of Register	1 - 5
1.5	Definition of Fill-In-the-Blanks and Format Image	1 - 6
1.5.1	Fill-In-the-Blanks	1 - 6
1.5.2	Format Image - Subformat Image	1 - 6
1.5.3	Image Page - Fill-In-the-Blanks Mask	1 - 6
2	FORMAT - IMAGE - SUBPROGRAM - TABLE CODING SHEETS	2 - 1
2.1	Format Coding Sheet and Image Coding Sheet	2 - 1
2.1.1	Format Coding Sheet	2 - 1
2.1.1.1	Subformat Head	2 - 1
2.1.1.2	Field Description	2 - 3
2.1.2	Image Coding Sheet	2 - 10
2.1.2.1	Subformat Head	2 - 10
2.1.2.2	Tag Description	2 - 10
2.2	Subprogram Coding Sheet	2 - 14
2.2.1	Subprogram Head	2 - 14
2.2.2	Subprogram Part	2 - 14

2.3 Table Coding Sheets	page 2 - 17
2.3.1 Single Entry Table Coding Sheets	2 - 17
2.3.1.1 Table Head With Argument Description	2 - 17
2.3.1.2 Argument Part	2 - 17
2.3.2 Double Entry Table Coding Sheet	2 - 19
2.3.2.1 Table Head With Argument Description and Function Description	2 - 19
2.3.2.2 Argument and Function	2 - 20
2.3.3 Disctable Coding Sheet	2 - 21
2.3.3.1 Table Head with Argument Description and Function Description	2 - 21
2.3.3.2 Argument and Function Part	2 - 23
3 THE FORMAT LANGUAGE	3 - 1
3.1 On Programming	3 - 1
3.1.1 What Is a Program?	3 - 1
3.1.2 The Elements of a Program	3 - 3
3.1.3 The Elements of the Format Language	3 - 4
3.1.4 Example In the Format Language	3 - 5
3.2 Character Set	3 - 9
3.2.1 Names	3 - 9
3.2.2 Arithmetic Operators	3 - 12
3.2.3 Relational Operators	3 - 12
3.2.4 Logical Operators	3 - 12
3.2.5 Punctuation Symbols	3 - 12
3.3 Operands	3 - 13
3.3.1 Constants	3 - 13
3.3.2 Registers	3 - 14
3.3.3 Fields	3 - 17
3.3.4 Subscripts	3 - 19
3.4 Notation	3 - 20
3.5 Arithmetic Expressions	3 - 21
3.6 Conditions	3 - 24
3.6.1 Relation	3 - 25
3.6.1.1 Comparison of Numeric Operands	3 - 26
3.6.1.2 Comparison of Nonnumeric Operands	3 - 26
3.6.2 Table Condition	3 - 28
3.6.3 Validity Condition	3 - 28
3.6.4 Compound Conditions	3 - 29

3. 7	Format Language Statements	page 3 - 31
3.7.1	Unconditional Statements	3 - 31
3.7.1.1	ALARM Statement	3 - 31
3.7.1.2	ALLOW and DISALLOW Statements	3 - 32
3.7.1.3	COMPUTE Statement	3 - 33
3.7.1.4	CONNECT Statement	3 - 34
3.7.1.5	DEFINE Statement	3 - 35
3.7.1.6	DISPLAY Statement	3 - 35
3.7.1.7	DUP Statement	3 - 36
3.7.1.8	END Statement	3 - 36
3.7.1.9	END SUBFORMAT Statement	3 - 37
3.7.1.10	GOTO Statement	3 - 37
3.7.1.11	LIMIT Statement	3 - 38
3.7.1.12	MOVE Statement	3 - 38
3.7.1.13	NOTE Statement	3 - 39
3.7.1.14	PERFORM Statement	3 - 40
3.7.1.15	SEARCH Statement	3 - 40
3.7.1.16	SELECT Statement	3 - 42
3.7.1.17	SET Statement	3 - 43
3.7.1.18	SKIP Statement	3 - 43
3.7.2	Conditional Statements	3 - 43
3.7.2.1	IF Statement	3 - 43
3. 8	Subprograms	3 - 45
3.8.1	Statements In Subprograms	3 - 45
3.8.2	Operands In Subprograms	3 - 46
4	EXECUTION OF FORMAT PROGRAMS	4 - 1
4.1	Selecting Subformat	4 - 1
4.2	Terminating a Format Program	4 - 1
4.3	Execution of Subformats	4 - 1
4.4	Execution of a Field Description	4 - 2
4.4.1	Keyed Fields	4 - 2
4.4.2	Automatic Fields	4 - 2
4.4.2.1	Duplicate Fields	4 - 2
4.4.2.2	Constant Fields	4 - 3
4.4.2.3	Increment Fields	4 - 3
4.4.3	Not Keyed Fields	4 - 4
4.4.4	Fields Skipped By SKIP	4 - 4
4.4.5	Fields Skipped By ENTER	4 - 4

4.4.6	Fields Skipped By RECORD RELEASE	page 4 - 4
4.4.6.1	Fields With Kind KEYED, DUPLICATE, CONSTANT, INCREMENT	4 - 4
4.4.6.2	Fields With Kind NOT KEYED	4 - 5
4.4.7	Fields Skipped By BYPASS	4 - 5
4.4.8	Execution of a Field Program	4 - 5
4.5	Field Flags	4 - 5
4.5.1	Validity Flag	4 - 6
4.5.2	Skipped Flag	4 - 6
4.5.3	Flags For REKEY	4 - 7
4.5.4	Flags For EDIT	4 - 7
4.6	Registers	4 - 7
4.7	Replay	4 - 8
4.8	Execution of IMAGE	4 - 8
5	ENTERING NEW FORMATS, SUBPROGRAMS, AND TABLES	5 - 1
5.1	New Formats	5 - 1
5.2	New Subprograms	5 - 9
5.3	New Tables	5 - 11
5.3.1	New Core Tables	5 - 11
5.3.2	Disc Tables	5 - 15
6	PROGRAMMING HINTS	6 - 1
6.1	Screen Processing	6 - 1
6.1.1	Screen Processing Assigned To the System	6 - 1
6.1.2	Establishing Keying Positions	6 - 2
6.1.3	Defining Tags	6 - 5
6.2	Reformatting	6 - 8
6.3	Automatic Insertion	6 - 9
6.3.1	Not Keyed Fields	6 - 9
6.3.2	Constant Fields	6 - 10
6.4	Automatic Duplication	6 - 11
6.5	Automatic Incrementation	6 - 13
6.6	The Use of Tables	6 - 14
6.7	Partial Rekeying	6 - 17
6.8	The use of Pseudo Registers	6 - 17

- APPENDIX I Required Space In Core For Formats,  
Subprograms and Tables
- APPENDIX II Required Space On Disc For Batches
- APPENDIX III Examples
- APPENDIX IV Standard Formats FORM, IMAGE, SUBPR,  
and TABLE
- APPENDIX V Format Language Syntax
- APPENDIX VI Limitations
- APPENDIX VII Messages from TRANSLATE
- APPENDIX VIII Definitions of Terms
- APPENDIX IX Index



# 0 Introduction

The RC 3600 Data Entry System is a software package operating under the RC 3600 Disc Operating Multiprogramming Utility System (DOMUS).

It is an input data preparation key-to-disc system, receiving data from local or remote key stations under format program control, and storing data on disc files. Whenever a data batch is completed, it may be dumped on tape or transmitted for remote processing. New format programs can be created and the formats are available to all key stations simultaneously.

The system offers a great variety of data manipulation possibilities during data entering, including: validity checking, rekeying, editing, skipping, duplication, arithmetic operations, batch accumulating, etc.

Supervisor functions include: format program generation, data batch transmission, etc.

The keying of new format programs is done under control of a standard format and the resulting format text is stored on the disc as a normal data batch. Now the supervisor may translate it to a format program and add it to the format library.

This manual contains a description of the format language, and instructions on the writing and translation of format programs.

As to the practical use of the RC 3600 Data Entry System, see User's Guide and Operating Guide.





# 1 Definitions

## 1.1 Definition of Batch – Record – Field 1.1

### 1.1.1 Batch 1.1.1

A batch is the area on a disc (a disc file), where the processed data are stored. The batch is output area for the processed document.

### 1.1.2 Record 1.1.2

A batch consists of a number of records. These records describe the logical structure of the processed document(s).

### 1.1.3 Field 1.1.3

A record contains a number of fields. A field is an element in the document that is processed as a single unit (e.g., a customer number, a name, an address).

#### Example 1

See Figure 1.1.3.

Here the document is an invoice and the corresponding batch may consist of a number of documents structured in the same way as this invoice.

Logically, the document may be divided into the following three parts:

A head, containing:

customer name, date, terms of payment, customer number, payment to, invoice number

An article line, containing:

article name, quantity, unit price, and final price

A total, containing:

total price

Each part corresponds to a record in the batch.

Figure 1.1.3



FRUIT MARKET Inc.

56, Orchard Road  
 APPLEVILLE  
 Phone: 076-33 44 11  
 Cables: fruitmark

Customer/Kunde:  
 L. Brown  
 39 Main Street  
 3000 Appleville

Delivered to/Leveret til:

Invoice date/Faktura dato: 76.09.03		Terms of payment/Betalingsbet.: c.o.d.			Date shipped/Forsend. dato:		Shipped by/Sendt med:	
Your ref./Deres ref.:		Our ref./Vor ref.: AA			Shipped from/Sendt fra:		Shipped to/Sendt til:	
Customers no./Kunde nr. 70950-0712	Serial no./Løbe nr.	Department no./Afdelings nr.:	Account type/Konto art:	Area no./Omr.nr.:	Flight no./Fly nr.:		A.W.B. no./Fragtbrev nr.:	
Payment to/Betaling til: Fruit Market Inc. 56 Orchard Road 3000 Appleville					Gross weight/Brutto vægt:		Net weight/Netto vægt:	
					Colli:		Country of origin/Oprindelsesland:	

INVOICE NO./FAKTURA NR. 0019908

ORIGINAL

Quantity/ Antal:		Unit price/ Stk. pris:	Amount/Beløb:
3	Apples	5.000	15.000
3	Pears	4.000	12.000
	Total		27.000

The individual columns in the document are viewed as separate elements, and each such element corresponds to a field in the batch.

## 1.2 Definition of Format, Subformat, and Field Description 1.2

### 1.2.1 Format 1.2.1

A format is a program for the Data Entry system. This program guides the keying of one or several given documents, and writes the formatted information to a batch on the disc.

### 1.2.2 Subformat 1.2.2

Each format is divided into a number of separate subformats. Each subformat controls the keying of one part of the document, possibly the whole document, and writes out the data, in formatted form, on the disc as one record.

Records produced with the same subformat have the same length, while records produced with different subformats may have different lengths.

### 1.2.3 Field Description 1.2.3

Each subformat consists of a number of field descriptions which describe the individual elements of a document.

Each field description controls the keying of one such element; calculating, reformatting, and writing it out as a field in a record.

### 1.2.4 Field Definition - Field Program 1.2.4

A field description consists of a field definition and a number of program statements which are referred to collectively as a field program.

If we return to the previous example (Figure 1.1.3), then a format for the keying of such a document could look like this:

FORMAT INVOI

SUBFORMAT 1:

customer	field description
date	field description
terms of payment	field description
customer no.	field description
payment to	field description
invoice no.	field description

SUBFORMAT 2:

article name	field description
quantity	field description
unit price	field description
final price	field description

SUBFORMAT 3:

total price	field description
-------------	-------------------

Now, if one uses

SUBFORMAT 1	once
SUBFORMAT 2	as many times as there are article lines in the document
SUBFORMAT 3	once

- then all filled-in columns in a document of this kind can be read and processed as fields in a batch.

## 1.3 Definition of Subprogram – Table

1.3

Special items in the format language are subprograms and tables. These are programmed and translated independently. They can be referenced by name from a format program or a subprogram.

### 1.3.1 Subprogram

1.3.1

A subprogram is a collection of program statements, which are executed when the subprogram is called ( from a format program or a subprogram ).

### 1.3.2 Table

1.3.2

A table is a collection of structured data which are referenced as one unit from a format program or a subprogram.

### 1.3.3 Argument - Function

1.3.3

A table is either single-entried or multiple-entried.

The 1st column is called argument;

The 2nd and following columns are called functions.

A single-entried table contains only one column (arguments).

A table consisting of two columns (argument and one function) is said to be double-entried.

Table data must be so structured that all columns are identical as to length and type. Examples on the use of tables are given in Section 6.6.

## 1.4 Definition of Register

1.4

Each format program can command a number of registers (X01-X99).

They can be used to transfer information from one subformat to another, or as working locations, or when transferring data to and from subprograms. The contents of user-defined registers can always be changed by a field program.

Registers may be of different length.

The format program can also access a number of pseudo-registers containing run-time information. These registers cannot be changed by the format program.

## 1.5 Definition of Fill-In-the-Blanks and Format Image 1.5

The image, or fill-in-the-blanks, facility is another special item contained in the system.

### 1.5.1 Fill-In-the-Blanks 1.5.1

Fill-in-the-blanks guidance assists while keying by displaying prompting messages on the screen. Let us look at example 1 again: a good guidance for keying subformat 2 will be the shown printouts. The place of the cursor will indicate the element to be keyed.

article name:	quantity:
unit price: _____	final price: _____
└──────────┘ blanks	└──────────┘ blanks for keying final price

Such printouts appear as fill-in-the-blanks (tags) on the screen. This means that there are blank spaces between the printouts and the specific columns are keyed in these blank spaces.

### 1.5.2 Format Image - Subformat Image 1.5.2

A format may have a format image attached. A format image is a program by which the fill-in-the-blanks are written to the operator during keying. The image is sectioned so that each subformat has its subformat image.

### 1.5.3 Image Page - Fill-In-the-Blanks Mask

A subformat image may be further divided into several image pages, each of which structures a fill-in-the-blanks mask, i.e., a screen image.

## 2 Format — Image — Subprogram — Table Coding Sheets

New formats, images, subprograms, and tables are coded on special coding sheets (documents), and the entering of these documents into the Data Entry system (by keying) is performed under format control, as with all other data. Batches created in this way are translated by calling the TRANSLATE supervisor program. After a correct translation the new format, subprogram, or table will be available for use in the system.

In the following the coding sheets will be presented column by column, with those columns that require keying marked with an asterisk (\*).

### 2.1 Format Coding Sheet and Image Coding Sheet 2.1

#### 2.1.1 Format Coding Sheet 2.1.1

The format coding sheet consists of two parts:

subformat head, and  
field descriptions.

See Figure 2.1.1

#### 2.1.1.1 Subformat Head 2.1.1.1

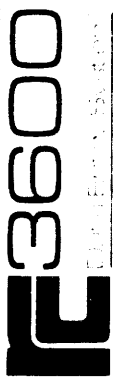
Column 1\*: Format name      Min. 1 character, max. 5 characters. The 1st character must be a letter, the following may be either letters or digits.

The format name identifies the current format in the system, and must differ from all existing names of formats, subprograms, tables, etc.

Column 2\*: Subformat name

1 character, letter or digit.

The subformat name must be unique within the format. The subformat name identifies the current subformat, meaning the subsequent row of field descriptions.



# FORMAT CODING SHEET

PAGE \_\_\_\_\_ OF \_\_\_\_\_

NOTES

INITIALS: \_\_\_\_\_ DATE: \_\_\_\_\_

FORMAT: \_\_\_\_\_

FIGURE 2.1.1

FORMAT NUMBER	COMMENT
1	4

LINE	FIELD	CHARACTER POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	FILE	KEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS		
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

7. TYPE = N, SN, SS, AN, A 10. FILL = \* 11. 12. 13. KIND = , N, C, D, I, K



Column 3: Protected

Indicates whether the current subformat is protected against manual selection or not.  
N = no protection, i.e., the subformat can be selected manually.  
Y = protection, i.e., the subformat can only be selected by the format (through a SELECT statement).  
If this column is empty, then N is understood.

Column 4: Comment

Min. 0 characters, max. 74 characters.  
A comment can be used, for example, to describe the format/subformat.

2.1.1.2 Field Description

2.1.1.2

Columns 1-14 constitute the field definition, while column 15 is a field program part.

Column 1: Field name

Min. 0 characters, max. 5 characters.  
If field name is specified, the first character must be a letter and the following either letters or digits.  
The field name identifies a field within the current subformat, and must be unique on subformat level.

The 2nd, 3rd, and 4th columns describe the current field's position on the screen (by indicating first field position).

Column 2: Page

Max. 1 digit; min. value = 1, max. value = 8.  
Used to divide a record into a number of parts (pages), each of which consists of a number of fields which together make up the screen image.  
The pages are numbered from 1 up. When page is indicated, its value must be either

equal to or greater than that of last indicated page number, and it must furthermore be accompanied by the indication of line and position (i.e., columns 3 and 4).

Column 3: Line

Max. 2 digits; min. value = 1, max. value = number of data lines on screen.  
Indicates on which data line the field is to be entered on the current page. When line is indicated, page and position must also be stated (= 2nd and 4th columns).

Column 4: Position

Max. 2 digits; min. value = 1, max. value = number of characters on screen line.  
Indicates position of first character of the field on current line, counting from left. The position number is limited so as to allow the whole field to fit into the remainder of the screen line.  
When position is indicated, page and line (= 2nd and 3rd columns) must also be stated.

If columns 2-4 are not keyed, one of the following will occur:

- If there is sufficient space left on current line: the field is placed after the preceding field, leaving a blank position in between.
- If there is not sufficient space left on current line: the field is placed on the next line, starting from the left-most position.
- If there is not sufficient space left on the current screen image: the field is placed on the 1st data line of the next page, starting in the left-most screen position.
- If the current field is the first field in the subformat: the field is placed on the 1st data line of the first page, starting in the left-most screen position.

Column 5(\*): Length

Max. 2 digits; min. value = 0, max. value = 80.

When length is greater than 0, the field length will be "length" = number of characters.

When length = 0, only the program part (column 15) of a field description can be stated. When using a format, no field input is required, but the program part of such a field will be executed.

When length is left blank, no other columns but the program part (= column 15) of the field can be stated, in which case this program part is treated as a continuation of the program part of the preceding field description.

Column 6(\*): Min. length

Max. 2 digits; min. value = 0, max. value = length (see 5th column) of current field description.

Indicates minimal number of characters to be keyed to the field; if min. length = 0, the field may be skipped. Indicated whenever length (column 5) > 0.

Column 7(\*): Type

Describes field type, i.e., which characters should be keyed to the field.

N = unsigned numeric.

Allowable characters:

1. Digits 0 through 9.
2. Fill characters.

The field will be treated as a positive expression.

SN = signed numeric.

Allowable characters:

1. Digits 0 through 9.
2. Minus sign (preceding first digit).
3. Fill characters.

When the minus sign is keyed, the field will be computed as a negative expression, otherwise as a positive. A minus sign is stored in its keyed position, and occupies thus a field position that would be free if no minus sign were keyed (leading separate representation).

SS = signed numeric.

Allowable characters:

1. Digits 0 through 9.
2. Fill characters.

When the field is terminated by the -ENTER key, it is treated as a negative expression; when the ENTER key is used, the field is assigned a positive value.

The negating operator is stored as an overpunch of the right-most character in the field - also called embedded trailing representation (0 becomes a ~, 1 becomes a J, 2 a K, 3 an L, etc.).

AN = alphanumeric.

Allowable characters: all non-control characters.

A = alphabetic.

Allowable characters:

1. Letters A through Z.
2. . , -
3. Fill characters.

The field is indicated when length (column 5) > 0.

Column 8 (\*): Output position

Max. 3 digits; min. value = 0, max. value = 255.

The position of the field in the output records is indicated by a field number, which permits reformatting the field sequence from input.

Fields with output position = 0 (no-transfer fields) are always placed after the last field in the output record. Such fields, though still stored in the output record (for possible rekeying) are not transferred by dump- or transfer programs.

The first field in the output record has output position = 1.

Only fields with a length (column 5) > 0 are counted.

Fields with length (column 5) = 0 are always placed after the last field in the output record.

Indication of output position is required if length (column 5) > 0.

Column 9: r/l

Indicates justification:

R = right-justified

L = left-justified

If the number of keyed characters is less than field length (column 5), the keyed characters are placed either in the right-most or in the left-most part of the field. Remaining positions are filled with fill characters (see next column specification!).

No indication = automatic right-justification.

Column 10: Fill characters

Specifies fill characters to fill not keyed positions in the field:

Δ = space

0 = zero

\* = asterisk

If fill character is not indicated, spaces are understood.

Column 11: Rekey

Indicates rekeying of a field:

Y = rekey field.

N = do not rekey field.

No indication implies rekeying.

Column 12: Display

Indicates whether an edited field shall be displayed on the screen or not. 'Editing' includes, among other things, justification and insertion of fill characters.

N = do not display edited field.

Y = display edited field (contents of output record field) justified and filled with fill characters.

Example: Display = Y may be used to show input of a not keyed field.

No indication of display implies N.

Column 13: Kind

Indicates field kind, that is:

K = keyed field. Field may be keyed.

N = not keyed field. No operator action required; the field contents may be computed by the field program.

C = constant field. Field contains either the contents of the register specified in column 14, or currently keyed field input.

D = duplication field. Either the field contains the value of the register specified in column 14, or one keys in the current value when the field is encountered. In the latter case the register is changed to the keyed value.

I = incrementation field. As for duplication field, except that - if no data are keyed in - the register value + 1 is entered to both field and register.

The field must be of type "N" (column 7).

When kind = C, D, or I, indication of register (=column 14) is required.

No indication of kind implies that kind = K.

Column 14: Register

Max. 2 digits; min. value = 1, max. value = 99.

Specifies which register should be used to hold the field contents if field kind = C, D, or I.

Register may only be specified for fields with kind (column 13) = C, D, or I.

Column 15: Program statements

Min. 0 characters, max. 80 characters.

Contains a part of a field program.

A field program consists of a number of these columns, which together form none, one, or several statements (see Section 3.7).

These statements are used when, for instance, submitting current field to closer control than what is specified in columns 5 through 14. If a statement is to include a reference to a field, the corresponding field name (as specified in its 1st column) must be indicated.

If there is not enough space in a column to include the whole field program, the field program may be continued in the next column 15, provided the preceding columns 1 through 14 are left empty.

A field program is considered to be concluded if a filled-in field description (columns 1 through 14) or a new subformat is encountered, or if the format is concluded.

If the format coding sheet is not large enough to hold the whole subformat, continue on a new format coding sheet, but leave the subformat head empty.

## 2.1.2 Image Coding Sheet

2.1.2

The image coding sheet consists of two parts:

subformat head, and  
tag descriptions.

See Figure 2.1.2.

### 2.1.2.1 Subformat Head

Column 1\*: Format name

Min. 1 character, max. 5 characters.

Format name must be identical to the name of the format where the current format image is used.

Column 2\*: Subformat name

1 character.

The subformat name must correspond to the name of a subformat within the format that uses those tags which are listed up to the appearance of the next subformat head or the end of the tag description.

Column 3: Comment

Min. 0 characters, max. 74 characters.

Comments are used to describe e.g. the screen layout.

### 2.1.2.2 Tag description

2.1.2.2

Together, the 1st, 2nd, and 3rd columns describe the screen position of the current tag (by indicating 1st text position).

Column 1\*: Page

Max. 1 digit; min. value = 1, max. value = 8.

The tags of one subformat are hereby divided into a number of parts (pages). Each page contains as many tags as together create one screen image.

The pages are numbered from 1 up. Tag descriptions belonging to the same page must appear in one sequence.







The page numbers are printed in unbroken, non-decreasing, sequence.

Column 2\*: Line

Max. 2 digits; min. value = 1, max. value = number of data lines on the screen. Indicates on which data line the actual text is to start (on current page).

Column 3\*: Position

Max. 2 digits; min. value = 1, max. value = number of characters on one screen line. Indicates position of the first character of current text on the current screen line. The position may not fill more than to allow the rest of the screen line to hold the whole text.

Column 4\*: Text

Min. 1 character, max. 80 characters. This contains the tag which is to be used by the current subformat in the screen position specified by page, line, and position (that is, the 1st, 2nd, and 3rd columns). The following spaces (i.e., superfluous spaces to the right of the text) are not included in the image.

## 2.2 Subprogram Coding Sheet

2.2

The subprogram coding sheet consists of two parts:

subprogram head, and  
subprogram parts.

See Figure 2.2.

### 2.2.1 Subprogram Head

2.2.1

Column 1\*: Subprogram name      Min. 1 character, max. 5 characters.  
1st character must be a letter, the following characters either letters or digits.  
A subprogram's name serves as its identification in the system and must differ from all existing formats, subprograms, tables, etc.

Column 2: Comment      Min. 0 characters, max. 74 characters.  
Comments may be used when, for instance, describing the subprogram.

### 2.2.2 Subprogram Part

2.2.2

Column 1: Program statement      Min. 1 character, max. 80 characters.  
Contains a part of a subprogram. A subprogram consists of a number of such columns, which together form one or more statements. (See Sections 3.7 and 3.8!)




	TABLE CODING SHEET (SINGLE)	PAGE	OF
	INITIALS:	DATE:	TABLE:
NOTES:			

FIGURE 2.3.1

TABLE NAME	T	A	A	V
1	2	3	4	5
S				
L	2	3	4	5
G	2	3	4	5
TH	2	3	4	5

T ARGUMENTS	
-------------	--

2.3 Table Coding Sheets 2.3

2.3.1 Single Entry Table Coding Sheets 2.3.1

A single entry table coding sheet consists of:  
table head with argument description, and  
argument part.

See Figure 2.3.1.

This coding sheet only applies to core-tables; coding of disc tables is performed on the disc table coding sheets (2.3.3)

2.3.1.1 Table Head With Argument Description 2.3.1.1

Column 1\*: Table name                      Min. 1 character, max. 5 characters.  
The first character must be a letter, the following either letters or digits. The table's name serves as its identification in the system, and must differ from all existing formats, subprograms, tables, etc.

Column 2\*: Type                                      = S, for Single entry table.

Column 3\*: A-type                                      This describes the argument type, thus:  
N = unsigned numeric  
AN = alphanumeric

Column 4\*: A-lgth                                      Max. 2 digits; min. value = 1, max. value = 80.  
Gives the argument length.  
All arguments have the same length.

2.3.1.2 Argument Part 2.3.1.2

Column 1\*: Argument                                      Min. 1 character, max. A-lgth characters.  
If A-type =  
N: Right-justify the argument and fill not keyed positions to the left of the argument with zeroes.  
AN: Left-justify the argument and fill not



# TABLE CODING SHEET (DOUBLE)

PAGE OF

INITIALS:

DATE:

NOTES:

TABLE:

FIGURE 2.3.2

TABLE NAME	T	A TYPE	A LGTH	F TYPE	F LGTH
1	2	3	4	5	6
	D				

1 & 2 ARGUMENTS & FUNCTIONS

1	
2	
1	
2	
1	
2	
1	
2	
1	
2	
1	
2	
1	
2	
1	
2	
1	
2	



keyed positions to the right of the argument with blanks (Δ).

An argument may not stretch over more than one line.

2.3.2 Double Entry Table Coding Sheet 2.3.2

A double entry table coding sheet consists of table head with argument description and function description, and argument part and function part.

See Figure 2.3.2.

This coding sheet only applies to core-tables; coding of disc tables is performed on the disc table coding sheets (2.3.3).

2.3.2.1 Table Head With Argument Description and Function Description 2.3.2.1

Column 1\*: Table name                      Min. 1 character, max. 5 characters.  
The first character must be a letter, and the following may be either letters or digits.  
The table name serves as its identification in the system and must differ from all existing names of formats, subprograms, tables, etc.

Column 2\*: Type                                = D, for Double entry table.

Column 3\*: A-type                            Describes the argument type, thus:  
N = unsigned numeric  
AN = alphanumeric

Column 4\*: A-lgth                            Max. 2 digits; min. value = 1, max. value = 80.  
Indicates the length of argument.  
All arguments have the same length.

Column 5\*: F-type                            Describes the function type, thus:  
N = unsigned numeric  
AN = alphanumeric

Column 6\*: F-lgth

Max. 2 digits; min. value = 1, max. value = 80.

Indicates the length of function.

All functions have the same length.

2.3.2.2 Argument and Function

2.3.2.2

Column 1\*: Argument

Min. 1 character, max. A-lgth characters.

If A-type =

N: Right-justify the argument and fill not keyed positions to the left of the argument with zeroes.

AN: Left-justify the argument and fill not keyed positions to the right of the argument with blanks ( $\Delta$ ).

An argument may not stretch over more than one line.

Column 2\*: Function

Min. 1 character, max. F-lgth characters.

If F-type =

N: Right-justify the function and fill not keyed positions to the left of the function with zeroes.

AN: Left-justify the function and fill not keyed positions to the right of the function with blanks ( $\Delta$ ).

A function may not stretch over more than one line.

2.3.3 Disc Table Coding Sheet. 2.3.3

A disc table coding sheet consists of table head with argument and function description, and argument part and function part.

See Figure 2.3.3.

2.3.3.1 Table Head with Argument Description and Function Description 2.3.3.1

Column 1* : Table name	Min. 1 character, max. 5 characters. The first character must be a letter, and the following may be either letters or digits. The table name serves as its identification in the system and must be the name of the disc table holding the table.
Column 2* : Type	= M, for possibly <u>M</u> ultiple entry table.
Column 7* : No. of Functions	A digit specifying the number of functions per argument; min. value = 0, max. value = 6. Zero means that the table is single-entried.
Column 8* : A - type	Describes the argument type, thus: N = unsigned numeric AN = alphanumeric
Column 9(*): F1 - type	Describes the type of first function, thus: N = unsigned numeric AN = alphanumeric Not filled in if no. of functions equals 0.
Column 10(*): F2 - type	Describes the type of second funtion (see column 9). Not filled in if no. of functions less than 2.
Column 11(*): F3 - type	Describes the type of third function (see column 9). Not filled in if no. of functions less than 3.
Column 12(*): F4 - type	Describes the type of fourth function (see column 9). Not filled in if no. of functions less than 4.



# DISC TABLE CODING SHEET

PAGE OF

INITIALS:

DATE:

NOTES:

TABLE:

Figure 2.3.3

TABLE NAME	T	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1															
M															

NO OF F'S	F1- TYPE	F2- TYPE	F3- TYPE	F4- TYPE	F5- TYPE	F6- TYPE
7	8	9	10	11	12	13
						14

F ONLY TO BE KEYED IF FILLED IN, OP = I, U, D, E.

OP ARGUMENTS & FUNCTIONS

1

2

Column 13(\*): F5 - type

Describes the type of fifth function (see column 9). Not filled in if no. of functions less than 5.

Column 14(\*): F6 - type

Describes the type of sixth function (see column 9). Not filled in if no. of functions less than 6.

2.3.3.2 Argument and Function Part

2.3.3.2

Column 1 (\*): Operation

Operation to be performed:

- I = Insert, the argument in column 2 on the same line and functions in following lines are to be inserted in the table as one entry.
- U = Uppdate, the functions in following lines replace the functions of an existing entry identified by the argument specified in column 2 of the same line.
- D = Delete, remove the entry identified by the argument specified in column 2 of the same line from the table.
- E = End, specifies that the line is the last line on the last coding sheet.

Column 2(\*): Arguments and Functions

If column 1 of the same line is filled in with an E then column 2 is empty.

If column 1 of the same line is filled in with a D then column 2 is an argument.

If column 1 of the same line is filled in with I or U then column 2 is an argument and the following lines contain the corresponding functions in column 2 (column 1 empty). The number of functions following the argument must correspond to the number specified in column 7 of the table head (see above).

Each argument and function must contain at least one and at most 80 characters, however, the lengths must correspond to the characteristics with which the disc table has been created, see Users Guide part 2.

An argument is specified as follows:

If A - type =

N: Right - justify the argument and fill not keyed positions to the left of the argument with zeroes.

AN: Left - justify the argument and fill not key positions to the right of the argument with blanks ( $\Delta$ ).

An argument may not stretch over more than one line.

A function is specified as follows:

If F<sub>x</sub> - type (x = function number) =

N: Right - justify the function and fill not keyed positions to the left of the function with zeroes.

AN: Left - justify the function and fill not keyed positions to the right of the function with blanks ( $\Delta$ ).

A function may not stretch over more than one line.

# 3 The Format Language

## 3.1 On Programming 3.1

### 3.1.1 What Is A Program? 3.1.1

A program can be viewed as the exact description of the procedure whereby you solve a specific problem.

Consider, for example, the problem of crossing a street without being overrun by a car. In a case like this it is not enough to know that you must "watch out before you cross the street", if you have not been confronted with precisely the same problem before.

Therefore, the problem must be analyzed, which means that one must try to survey the parameters contained in the problem, and to assess their different roles therein.

In order to be able to cross the street you must therefore know that a car might come, that it might come from left or right, and that it might prevent your getting across the street.

Thus, a program must be a step by step description of how the parameters (operands) contained in a problem should be handled so as to arrive at the desired final stage from a given starting point.

In the example of crossing a street one may choose as a starting point the situation where the program ignores the events leading up to that situation. As the final stage one selects the arrival at the opposite sidewalk. Written in ordinary language, such a program might look something like this:

Example 3.1.1a

1. Look to the left.
2. Do you see a car?  
Yes: Go to point 3.  
No: Clear, go to point 5.
3. Is the car less than 200 meters away?  
Yes: Go to point 4.  
No: Clear, go to point 5.
4. Is the car parked?  
Yes: Clear, go to point 5.  
No: Go to point 1.
5. Look to the right.
6. Do you see a car?  
Yes: Go to point 7.  
No: Clear, go to point 9.
7. Is the car less than 200 meters away?  
Yes: Go to point 8.  
No: Clear, go to point 9.
8. Is the car parked?  
Yes: Clear, go to point 9.  
No: Go to point 1.
9. Walk to the opposite sidewalk.

As suggested above, one could use another, preceding, program to describe how to reach the specific street that one is to cross, and a following program to specify what actions to take once one has crossed the street.

Therefore, the starting situation could be altered a little. Let us say that,

1. You are standing at the curbline, and
2. You know that the street has two-way traffic or, if not, that it is a one-way street (with traffic from left or from right).

The source of your information may be a previously executed program (its final stage), and in rewriting the program given in the example above (3.1.1a) you can insert the information under 2., so as to be able to decide which way to look:



### Example 3.1.1b

1. Does the street have two-way traffic?  
Yes: Go to point 1b.  
No: Go to point 1a.
- 1a. Does the street have one-way traffic from the left?  
Yes: Go to point 1b.  
No: One-way, from the right: Go to point 5a.
- 1b. Look to the left.  
.  
.  
.
5. Does the street have one-way traffic only?  
Yes: Clear from the left, go to point 9.  
No: Go to point 5a.
- 5a. Look to the right.  
.  
.  
.

We could further extend the example's program. It could, for example, count all cars passing from the right, it could register how many times one had looked to the left before the street was safe to cross, and so on. This information could be fed into a following program, e.g., for statistical use.

All information that is available for a program in the starting situation is called input parameters, which, together with the program's own calculations, may influence the execution of the program. Information derived from a program is called output parameters.

### 3.1.2 The Elements of a Program

3.1.2

A program consists of a number of statements. Every single statement's execution marks a step on the way from the program's starting situation to its final stage. The statements are written in a programming language, that is characterized by the firm rules that guide the formulation of a statement.

In the examples 3.1.1a and 3.1.1b every step can be viewed as a program statement.

The statements operate on a set of parameters (operands) that can be read and changed.

In Example 3.1.1b point 1 can be viewed as the read-out of the input parameter, stating the direction of traffic. If the example had been extended also to register the number of passing cars, this number would be a parameter that would be changed during the course of the program, from zero at the starting situation to the actual number of cars at the final stage.

The execution of the program begins with the execution of its first statement. This done, the next statement is executed, and so forth, until the program's last statement has been reached.

As can be seen from the examples above, the statements of a program are not executed in unbroken order; some statements are skipped as a result of the answers that are given to the questions presented.

### 3.1.3 The Elements of the Format Language

3.1.3

There are two types of programs in the Format Language: field programs and subprograms. A field program consists of those program statements that belong to a field description. A subprogram is a labelled collection of program statements, which are executed when referenced from a program. Subprograms are used if the repeating of the same sequence of statements in several programs is to be avoided.

The operands for a program are called variables and constants.

A variable is a place for storing information that can later be called upon and may be subject to change. In the Format Language, the variables are fields and registers. A variable is referenced from a program statement by calling it by name. A field is referenced by using the name of a field description, by which the program is made capable of processing a keyed unit of a specific part of a document. Fields cannot be referenced from subprograms.

A register is referenced by putting an "X" before its number. Registers in a given format are used to transfer information between programs, and to store intermediate results within programs.

Variables may contain different kinds of values, e.g., numerical or alphabetical.

A constant contains information that cannot be changed by the program, but is entered into calculations together with variables. Like variables, constants may contain different kinds of values. In the Format Language several constants may be assembled into a specified table, and can thus be referenced collectively from the program.

3.1.4 Example In the Format Language

3.1.4

Problem:

A document contains, among other things, the following elements:

DATE OF PURCHASE	□□□□	DATE OF PAYMENT	□□□□	AMOUNT:	□□□□
------------------	------	-----------------	------	---------	------

The two dates are specified by year and month (format YYMM). Date of purchase may not come later than date of payment. If purchase date coincides with date of payment, the program calculates a discount.

Problem analysis, program planning:

The first field program controls the date of purchase, and uses as input parameter a field with a keyed four digit number (specified in the field program's definition section).

The second field program controls the date of payment in the same manner as the first field program; it further compares the two dates. Thus the date of purchase will be one of the program's input parameters.

Since these dates are identically checked, one can use a subprogram, with the date itself as the input parameter and a correct/not correct indication as the output parameter. Month and year are separately checked; the year must be within the 70-80 (inclusive) interval, which is used as a constant.

As for the amounts, these are not subject to special verification, and a corresponding field description therefore involves only the definition section.

Calculation of discounts is performed by the field program of a 'not-keyed field' (meaning a field which receives its value solely from the calculating done by the program). As input parameters the program uses the two dates and the amount, and the resulting discount amount will appear in the field as the output parameter. The discount percentage is a constant.

Example 3.1.4a shows how to write field descriptions, and in Example 3.1.4b you will find the subprogram that checks the date.

The program statements will be discussed later in this manual. Here only the following information is given:

- 'DEFINE X01 4' defines a register of length 4.
- 'COMPUTE X01 = KDATE' transfers the contents of a field to a register.
- 'PERFORM DCHEK' causes the subprogram DCHEK to be executed, after which the program continues with the next statement.
- 'ALARM 'DATE OF PURCHASE WRONG'' causes the writing of an error message, whereupon the information must be keyed anew, and the field program is again executed from the beginning.
- 'IF KDATE > BDATE', > means 'IF GREATER THAN'.
- 'DISCOUNT = AMOUNT/100\*3' places the result of an arithmetical expression in a field. The discount rate is 3 percent of the amount.



# FORMAT CODING SHEET

PAGE  OF

FORMAT:

DATE:

INITIALS:

NOTES:

EXAMPLE 3.1.4a

FORMAT NAME	S	P	COMMENT
1	2	3	4

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN. LENGTH	TYPE	OUTPUT POSITION	R.L.	FILL	REKEY	DISPLAY	REGISTER	PROGRAM STATEMENTS
KDATE			0	4	4	N	10						DEFINE X01 4, DEFINE X02 2, COMPUTE X01 = KDATE, PERFORM DCHEK, IF X01 = 0 THEN ALARM 'DATE OF PURCHASE WRONG', COMPUTE X01 = BDATE, PERFORM DCHEK, IF X01 = 0 THEN ALARM 'DATE OF PAYMENT WRONG', IF KDATE > BDATE THEN ALARM 'DATE OF PURCHASE GREATER THAN DATE OF PAYMENT', IF KDATE = BDATE THEN COMPUTE DISCOUNT = AMOUNT/100*3 ELSE COMPUTE DISCOUNT = 0,
BDATE			4	4	4	N	11						
AMOUN			10	1	1	N	12	0					
DISCO			9	9	9	N	13	0					

INITIALS:

DATE:

NOTES:

SUBPROGRAM:

EXAMPLE 3.1.4b

SUB-PROG NAME	COMMENT
1	2
DCHEK	START: X01 = YYMM, STOP: X01=0 WHEN WRONG DATE, X02 IS WORKING REGISTER

PROGRAM STATEMENTS

```
1 COMPUTE X02 = X01 MOD 100, NOTE X02 = MM,  
2 COMPUTE X01 = X01 / 100 , NOTE X01 = YY,  
3 IF (X01 < 70) OR (X01 > 80) THEN GOTO NOTOK,  
4 IF (X02 < 1) OR (X02 > 12) THEN GOTO NOTOK,  
5 NOTE THE DATE IS CORRECT,  
6 COMPUTE X01 = 1,  
7 GOTO STOP,
```

NOTOK:

```
1 COMPUTE X01 = 0,
```

STOP:

END,

The character set for the format language consists of 50 characters. These characters and their corresponding meanings are:

<u>Character</u>	<u>Meaning</u>
0, 1, 2, 3, 4, 5, 6, 7, 8, 9	Digit
A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z	Letter
Δ	Space
+	Plus sign
-	Minus sign or hyphen
*	Asterisk
/	Stroke
=	Equal
,	Comma
;	Semicolon
'	Quotation
(	Left parenthesis
)	Right parenthesis
>	Greater than
<	Less than
:	Colon

The basic elements of the language are:

names, arithmetic operators, relational operators, logical operators, and punctuation symbols.

The elements are explained in the following sections.

### 3.2.1 Names

3.2.1

A name is composed of a combination of characters. Allowable characters are:

Letters: A through Z

Digits: 0 through 9

A name must begin with a letter. Only the five leading characters are significant. Thus PERFORM is equivalent to PERFO, for example, and PERFO is equivalent to PERFORMANCE; but IN is not equivalent to INCORRECT.

There are 6 types of names: reserved names, user-defined field names, user-defined label names, user-defined subprogram names, user-defined table names, and user-defined subformat names.

Reserved names have a special predefined meaning to the system; therefore, these must never be used as user-defined names. Reserved names are listed in Table 3.2.1-1.

Some of the reserved names are verbs. Verbs identify statements in the format language and are used in field programs and subprograms. The verbs allowed in each of the two programs are listed in Table 3.2.1-2.

Table 3.2.1-1. List of Reserved Names

ALARM	GIVING	SET
ALLOW	GOTO	SKIP
ALPHANUMERIC	IF	SUBFORMAT
AND	IN	THEN
AT	INVALID	TO
COMPUTE	LIMIT	VALID
CONNECT	MOD	X00
DEFINE	MOVE	X01
DISALLOW	NOT	:
DISC	NOTE	X99
DISPLAY	NUMERIC	XBATCH
DUP	OR	XDATE
ELSE	PERFORM	XJOB
END	SEARCH	XOPERATOR
FIELD	SELECT	XTIME



Table 3.2.1-2. List of Reserved Verbs

<u>Verb</u>	may be used in:	
	<u>Field Program</u>	<u>Subprogram</u>
ALARM	x	x
ALLOW	x	x
COMPUTE	x	x
CONNECT	x	x
DEFINE	x	x
DISALLOW	x	x
DISPLAY	x	x
DUP	x	x
END	x	x
END SUBFORMAT	x	
GOTO	x	x
IF	x	x
LIMIT	x	x
MOVE	x	x
NOTE	x	x
PERFORM	x	x
SEARCH	x	x
SELECT SUBFORMAT	x	
SET	x	
SKIP	x	x

### 3.2.2 Arithmetic Operators

3.2.2

The arithmetic operators are used to perform specific arithmetic operations.

The used symbols and their operation are :

+	Addition
-	Subtraction
*	Multiplication
/	Division
MOD	Modulo
NUMERIC	Conversion of nonnumeric to numeric
ALPHANUMERIC	Conversion of numeric to nonnumeric

### 3.2.3 Relational Operators

3.2.3

Relational operators specify the type of comparisons to be made between two operands in relational conditions. These symbols and their meaning are :

>	Greater than
>=	Greater than or equal to
=	Equal to
<	Less than
<=	Less than or equal to
<>	Not equal to

### 3.2.4 Logical Operators

3.2.4

Logical operators are reserved names that define a connection between operands. The reserved names and their use are :

AND	Logical 'and'
OR	Logical 'or'
NOT	Logical 'not'

### 3.2.5 Punctuation Symbols

3.2.5

The punctuation symbols used in the program statement section of a format program, and their names, are :

Δ	Space	(	Left parenthesis
,	Comma	)	Right parenthesis
;	Semicolon	'	Quotation mark
:	Colon		

The operands used in the format language are described in the following paragraphs. The operands are:

Constants

Registers

Fields

Registers and fields may be used with subscripts; this feature is described in section 3.3.4.

Operands may be used as destination or as source, except constants which only may be used as source. Source means that the operand is 'input parameter' to a statement. Destination means that the operand is 'output parameter' from a statement.

Operands may be numeric or nonnumeric. A numeric operand contains a numeric value. A nonnumeric operand contains a string of characters.

#### 3.3.1 Constants

#### 3.3.1

Constants are strings of characters which represent a specific value. There are two types of constants: numeric and nonnumeric.

A numeric constant is composed of digits, and must contain at least one digit but not more than 80 digits. The value of a numeric constant is always positive. Negative values are obtained in the statements, where it is allowed, by preceding the numeric constant by a minus.

Examples of valid numeric constants are:

198

50

091

Examples of invalid numeric constants are:

-198	Sign is not allowed
1.5	Cannot contain a decimal point
9,85	No comma allowed

A nonnumeric constant can contain any characters including those not in the format language character set, except quotation marks. The constant must be enclosed in quotation marks. A nonnumeric constant can be from 0 to 78 characters.

Examples of valid nonnumeric constants:

```
'MONTH IS GREATER THAN 12'  
'19876'  
'TYPE N FOR NO, Y FOR YES'  
"
```

Example of invalid nonnumeric constant:

```
'TYPE 'N' FOR NO'      Cannot contain quotation marks
```

A nonnumeric constant may look like a numeric constant, but the two are not identical. They are both stored as characters, but a numeric constant is interpreted as a numeric value. Thus the numeric constant 00190 is equivalent to the numeric constant 190, but the nonnumeric constant '00190' is not equivalent to the nonnumeric constant '190' or ' $\Delta\Delta$ 190'.

### 3.3.2 Registers

3.3.2

Registers have been added to extend the possibilities of the language. Registers are normally used in three connections. They are used for transferring data from one record to another, for transferring data to and from a subprogram and for computations. Registers are defined by the DEFINE statement, see Section 3.7.1.5.

The letter X followed by any number from 01 to 99 is used to name a register. As many registers as needed can be used in a format program. The numbers used need not be sequential, but should be sequential starting with 01 to conserve storage.

Examples of valid register names:

X01

X11

X99

Examples of invalid register names:

X00        Digits not 01 to 99

X1         Too few digits

X010      Too many digits

A register contains either numeric or nonnumeric data. The type of data is dependent of the program statement, which had the register as destination last time (e.g., the MOVE statement makes the register nonnumeric, the COMPUTE statement numeric). Data are always stored as characters in the register. The length of a register is declared by a DEFINE statement and is the number of character positions in the register. The DEFINE statement must be executed before any other statement referring to that register. The DEFINE statement gives no type to the register; the type is given first when the register is used as destination in a statement. It is allowed to change the type of a register during execution of the format program.

If nonnumeric data are stored in a register, and the number of characters in the data is smaller than the register length, data are stored from left to right and the remaining positions in the register are filled with spaces. If the number of characters in the data is greater than the register length the right-most characters are truncated.

Numeric data are right-justified in the register, and remaining positions are filled with zeroes if the data are created by the COMPUTE statement. The negating operator of data representing a negative value will be stored as for type SS (see 2.1.1.2). If the number of significant digits in the data is greater than the register length, a runtime error will occur.

Numeric data stored automatically in a register (automatic fields, see 4.4.2) will always be identical to the data stored in the field also with respect to justification, fill characters and representation of negating operator.

An automatic field, of which the length of the register unequals the length of the field, will cause a runtime error when the field is encountered.

The system furthermore contains a number of pseudo-registers containing run-time information which may be accessed by programs.

The pseudo-registers are predefined by the system their names belonging to the reserved names of the language. This means that these registers may be accessed without having been defined by DEFINE statements in the format program, and that they do not claim storage to be used.

The contents of the pseudo-registers cannot be altered by the programs which means that they cannot occur:

- on the left side of the equal sign in COMPUTE statements (see 3.7.1.3).
- after the word GIVING in CONNECT and SEARCH statements (see 3.7.1.4 and 3.7.1.14)
- after the word TO in MOVE statements (see 3.7.1.11).
- in field definitions (see 2.1.1.2, column 14: register).

Apart from these limitations the pseudo-registers may be used in all connections where registers are allowed as source operands. The operand type of the pseudo-registers is nonnumeric.

The pseudo-registers are:

name	length in characters	type	contents
XBATCH	5	AN	Name of the batch in which processed data are stored.
XDATE	8	AN	Year, month, and day in the format: YY.MM.DD corresponding to the time at which a readout of the register occurs.
XJOB	5	AN	Name of the job to which the batch is belonging.
XOPERATOR	3	AN	Operator initials of key station operator.
XTIME	8	AN	Hour, minute, and second in the format: HH.MM.SS corresponding to the time at which a readout of the register occurs.

Operator initials are the initials given by the key station operator when work on a key station is initiated. For further details, see Users Guide, Part 1, which also describes when and how name of batch and job are entered by the keying operator.

### 3.3.3 Fields

3.3.3

Fields are numeric or nonnumeric depending on their type as specified in the field definition:

<u>Field Definition Type</u>	<u>Operand Type</u>
N	numeric
SN	numeric
SS	numeric
AN	nonnumeric
A	nonnumeric

Current field is the field which field description contains the field program.

Any field before (in the same subformat) and including current field is allowed as source operand in a field program. No field, except current field, is allowed as destination operand and current field is only allowed as destination if it is not keyed (i.e., kind = N).

The following rules apply to not keyed fields used as destination operands:

1. It is only allowed to store nonnumeric data in fields of type A or AN, and numeric data in fields of type N, SN, or SS.
2. When nonnumeric data are stored and the number of characters in the data is smaller than the field length, the data are left- or right-justified depending on the specification in the field definition, and the remaining positions are filled with the fill character specified in the field definition.

If the number of characters in the data is greater than the field length, the right-most characters are truncated. Then it is checked whether the field contents correspond with the type assigned in the field definition, otherwise a runtime error will occur (only applies for type = A).

Examples of storing nonnumeric data:

Field length	Type	Fill	Justi- fication	Source	Field contents
6	A	Δ	L	'ABCΔΔΔ'	'ABCΔΔΔ'
4	A	Δ	L	'ADDRESS'	'ADDR'
6	A	Δ	L	'ABC'	'ABCΔΔΔ'
6	A	Δ	R	'ABC'	'ΔΔΔABC'
6	A	Δ	L	'123'	runtime error
3	AN	Δ	R	'JANUARY'	'JAN'
10	AN	Δ	R	'JANUARY'	'ΔΔΔJANUARY'
6	AN	Δ	L	'123'	'123ΔΔΔ'

3. When numeric data are stored and the number of significant digits (leading zeroes are ignored) is smaller than the field length, the data are left- or right-justified depending on the specification in the field definition, and the remaining positions are filled with the fill character specified in the field definition.

If the number of significant digits is greater than the field length a runtime error will occur. It is then checked whether the field contents correspond with the assigned type in the field definition, otherwise a runtime error will occur (only applies for type = N).

Examples of storing numeric data:

Field length	Type	Fill	Justifi- cation	Source	Field contents
5	N	0	R	155	00155
5	N	0	R	-50	runtime error
5	N	0	R	555555	runtime error
5	SN	Δ	R	-50	Δ-50
5	SN	Δ	R	-50000	runtime error
5	SN	0	L	-50	-5000
5	SS	0	R	59	00059
5	SS	0	R	-55555	5555n
5	SS	0	R	-51	0005j



Subscripts are used to refer to individual characters or groups of characters in a register or a field.

The syntax of subscripts is (notation is described in 3.4):

numeric constant [ : numeric constant ]
---

A single character is accessed by supplying one subscript. A group of characters is accessed by supplying the leftmost subscript followed by the rightmost subscript of a sequence of adjacent characters in a register or field. A group of characters consists of two or more characters.

Subscripts should be in the range 1 to the length of the register or the field, where 1 is the leftmost subscript.

Examples of subscripting are:

```
MOVE XDATE(4:5) TO MONTH,  
COMPUTE A = B(3) * B(5),  
COMPUTE X01(5) = 9,  
MOVE 'DD' TO X03(7:8),
```

If at run-time, the value of the subscript exceeds the size of the register or the field being subscripted, a runtime error will occur.

When destination operands are supplied with subscripts, please notice the following rules:

1. Only the contents of the character position (s) specified by the subscript is changed. The justification and filling described in the preceding sections are not executed.
2. A negative value cannot be assigned to a subscripted numeric operand.
3. A register must be initialized before it is used as destination with subscript. The initialization may be performed by any statement which has the register as destination. The initialization is necessary because when registers are used as destination with subscript, it is required that the type of the source and the type of the register concur.

Example: If a register is used nonnumerically it may be initialized with a MOVE statement.

```
MOVE '' TO X01
```

and then used with subscript, e.g.,

```
MOVE 'A' TO X01(1),  
MOVE 'Z' TO X01(10),
```

If a register is used numerically it may be initialized with a COMPUTE statement:

```
COMPUTE X02 = 0
```

and then used with subscript, e.g.,

```
COMPUTE X02(1) = 5,  
COMPUTE X02(3) = 9,
```

4. It is not necessary to initialize a field before it is used as destination, because a not keyed field is always filled with the specified fill character before a field program is executed.
5. After a field is used as destination with subscript, the contents of the field are checked against the type of the field (only applies to type A), and a runtime error will occur if the contents and the type do not correspond.

### 3.4 Notation

3.4

The notation used in the remainder of this section is described in the following paragraphs:

1. All words printed in capital letters belong to the language. They are referred to as 'reserved names'.
2. Variable entries which are to be supplied by the format programmer are printed in lower case letters.
3. When punctuation or other special characters are printed, they are required.

4. Braces { } enclosing vertically listed items indicate that one and only one of the items is required.
5. Brackets [ ] are used to enclose a portion which is optional.
6. The ellipsis ... indicates that the preceding entity can occur one or more times in succession.

### 3.5 Arithmetic Expressions

3.5

Arithmetic expressions (or shortly: expressions) are used in certain program statements (the IF and COMPUTE statements, see Section 3.7). An arithmetic expression is composed of operands, parentheses and arithmetic operators according to certain rules which make an expression written almost as in the mathematical literature.

A simple example of a statement containing an arithmetic expression is:

```
COMPUTE X01 = X01 + X02,
```

where 'X01 + X02' is the expression the evaluation of it being the sum of values of register X01 and register X02.

The following rules concerning the type of result of an arithmetic expression must be observed:

1. The evaluation of an arithmetic expression in COMPUTE statements must result in a numeric value.
2. A relation (see 3.6.1) may compare either two arithmetic expressions giving nonnumeric values or two arithmetic expressions giving numeric values.

The arithmetic operators allowed are:

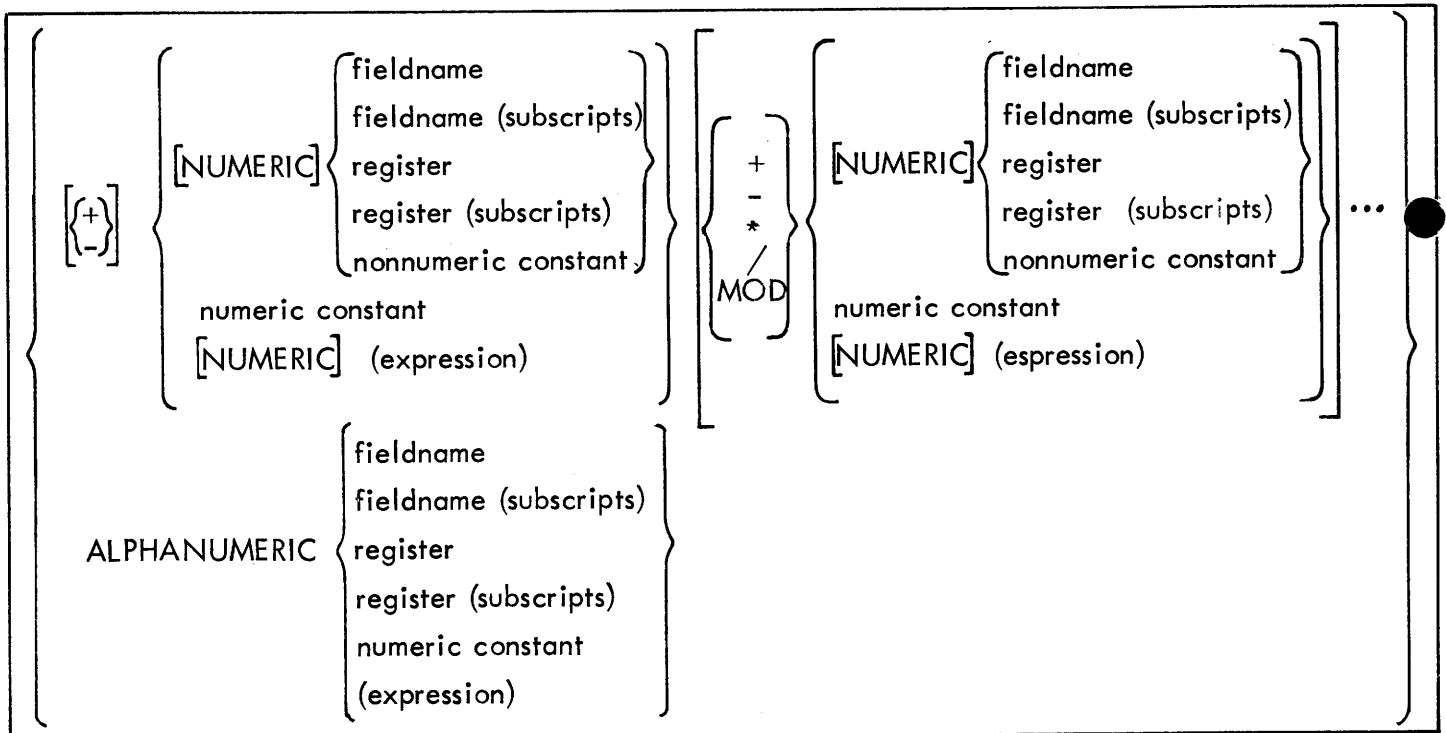
<u>Operator</u>	<u>Meaning</u>	
+	addition	} called adding operators
-	subtraction	
*	multiplication	} called multiplying operators
/	division	
MOD	modulo	
ALPHANUMERIC	convert numeric to nonnumeric	} called conversion operators
NUMERIC	convert nonnumeric to numeric	

The most simple arithmetic expression consists of merely one numeric operand.

More complex arithmetic expressions may be composed by:

1. separating two or more operands by one of the arithmetic operators;
2. preceding one operand with one of the adding operators (e.g., COMPUTE X01 = +5, COMPUTE X01 = -X01);
3. Subexpressions, enclosed in parentheses, may be used as an operand.

The rules for composing arithmetic expressions are:



The following rules concerning conversion operators must be obeyed:

1. The entity (fieldname, register, constant, or parenthesed expression) following the NUMERIC operator must be of nonnumeric type. An operand to be converted by NUMERIC must only contain digits and fill characters and the conversion cannot result in a negative value (negating operator not allowed).
2. The entity following the ALPHANUMERIC operator must be of numeric type.

Examples of arithmetic expressions are:

```
X01
FLD1
59
+X02(1)
-59
NUMERIC X10
ALPHANUMERIC FLD5
X01*FLD
X04*NUMERIC XDATE(7:8)
X01(1)+X01(2)+X01(3)
(X01+X02)/2
ALPHANUMERIC (NUMERIC X06*3)
FLD1 MOD 10 + FLD2 MOD 10
(FLD1(1)*2+FLD1(2)*3)/(FLD(1)+FLD(2))
((A+B) * (C+D) + (A+B)/2) MOD 10
```

An arithmetic expression is evaluated in the following order:

- 1 (first): Subexpression in parentheses
- 2 : Conversion operators (ALPHANUMERIC, NUMERIC)
- 3 : Multiplying operators (\*, /, and MOD)
- 4 (last): Adding operators (+ and -)

When a sequence of operators has the same priority, the operators are executed in order of their occurrence from left to right.

An example of evaluating an arithmetic expression:

Consider the expression:

$$-(A+B) * (C-D) / 2$$

First the subexpressions in parentheses are evaluated. A is added to B giving a temporary result, namely R1, and D is subtracted from C giving another temporary result R2. The expression may now be shown as:

$$-R1 * R2 / 2$$

Now the multiplying operators are executed in order of their occurrence from left to right, therefore R1 is multiplied to R2 giving the temporary result R3, and the expression may be shown as:

$$-R3 / 2$$

The second multiplying operator is executed: R3 is divided by 2 giving R4. Finally, R4 is negated, and the evaluation is completed.

The following examples show the evaluation of divisions:

<u>expression</u>	<u>result</u>	<u>expression</u>	<u>result</u>
7/3	2	7 MOD 3	1
7/(-3)	- 2	7 MOD (-3)	1
(-7)/3	- 2	(- 7) MOD 3	- 1
(-7/(-3))	2	(- 7) MOD (-3)	- 1

## 3.6 Conditions

3.6

Conditions are used in the so-called conditional statements (the IF statement, see Section 3.7). A condition causes the path of control to be altered depending upon whether the condition is true or false. A simple example is the following statement:

IF TOTAL < X01 THEN COMPUTE X02 = X02 + 1,

The condition is 'TOTAL < X01'. If TOTAL is less than X01, the condition is true and the new value of X02 is computed. If TOTAL is not less than X01, the condition is false and control is transferred to the next statement following the IF statement.

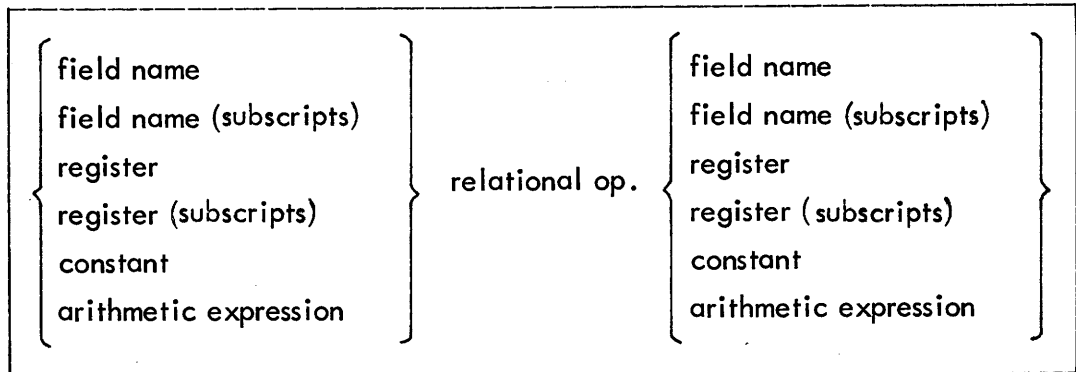
In the format language the following types of conditions are allowed: relations, validity conditions, and table conditions.

A condition may be either simple or compound. A simple condition is a relation, a validity condition or a table condition. A compound condition is composed of conditions, parantheses, and logical operators.

A relation is a comparison of two operands, either of which can be a field name, a register, a constant, or an arithmetic expression. The operands are separated by a relational operator which specifies the type of comparison to be made between the two operands. The allowable relational operators and their corresponding meanings are:

<u>Relational operator</u>	<u>Meaning</u>
>	greater than
>=	greater than or equal to
=	equal to
<=	less than or equal to
<	less than
<>	not equal to

The syntax of a relation is:



The following rules apply to relational conditions:

1. A numeric operand can only be compared with another numeric operand.
2. A nonnumeric operand can only be compared with another nonnumeric operand.
3. Comparisons of numeric and nonnumeric operands are not allowed. When needed, however, the conversion operators NUMERIC and ALPHANUMERIC may be used to convert the type of an operand.

Examples of relations:

```

X01 >= 0
X01 = X02
X01 * 2 - A <= X02
D = 'JENSEN'
(A + B) > 3

```

X05 = 'TEXT'  
X03(1) \* 2 >= 10  
(A(1) + A(2)) MOD 10 = 0  
ALPHANUMERIC FLD = '\*\*\*\*\*'  
YEAR <= XDATE(1:2)

3.6.1.1 Comparison of Numeric Operands

3.6.1.1

If the operands are numeric the respective values of the operands are compared.

3.6.1.2 Comparison of Nonnumeric Operands

3.6.1.2

The characters used in nonnumeric operands are ordered according to their position in a sequence of (all) characters. The relation between two characters is determined by their positions in the sequence of characters. The character sequence in ascending order is:

(space)  
!  
#  
\$  
%  
&  
'  
(  
)  
\*  
+  
,  
-  
.  
/  
0 through 9  
:  
;  
<  
=  
>  
?  
@  
A through Z  
↑  
~



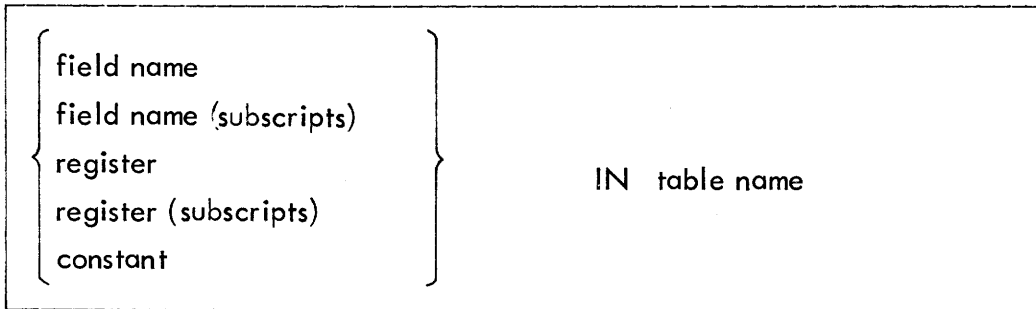
When nonnumeric operands are of equal length (i.e., they contain the same number of character positions), characters in corresponding positions of the two operands are compared starting with the left-most position and proceeding to the right-most position. If all the characters are the same through the last position, the operands are considered equal. If a pair of unequal characters is encountered, the position in the character sequence is determined for each character. The operand containing the highest character position is considered to be the greatest of the two operands. See the examples below!

When nonnumeric operands are of unequal length (i.e., they do not contain the same number of character positions), the longest operand is treated as if the right-most characters were truncated, to make it the same length as the other operand. The comparison is then made as though they were the same length.

Examples of comparing two nonnumeric operands A and B:

A	B	TRUE RELATION	EXPLANATION
'JENSEN'	'HANSEN'	$A > B$	Operands are of equal length. Characters are compared from left to right. J comes after H in the character sequence.
'FIELD'	'FIELDS'	$A = B$	The right-most character is truncated in B.
'29'	'199'	$A > B$	The right-most character is truncated in B, and 2 follows 1 in the character sequence.
'Δ 39'	'029'	$A < B$	0 comes after space in the character sequence.
'HANSEN'	'HANSON'	$A < B$	Characters are compared from left to right, the first pair of unequal characters is E and O, and E is preceding O in the character sequence.

A table condition is used to search a table for a specific argument. The syntax of a table condition is:



The table name identifies the table in the library of tables used. The table may be either a single-entried or a double-entried core table, but not a DISC table (see Section 5.3).

The arguments in the table, specified by table name, are searched for a match against the operand preceding the word IN. If a match is found the condition is true, otherwise the condition is false.

The operand preceding IN and the arguments in the table must be of the same type (i.e., either numeric or nonnumeric). The methods for comparing operands as described in Sections 3.6.1.1 and 3.6.1.2 are also used when evaluating a table condition.

Examples of table conditions:

```
X01 IN TABL1
FLD IN CTABL
FLD(1) IN CTABL
```

The validity condition determines whether a field is valid or invalid. The syntax of a validity condition is:



Every field has an associated validity flag, which can be explicitly set to valid or invalid by the SET statement (see Section 3.7.1.16). Unless changed by a SET statement, the validity flag of a field is invalid if an error has been detected and not corrected, otherwise the validity flag is valid.

### 3.6.4 Compound Conditions

3.6.4

Conditions, parentheses, and logical operators may be combined to form a compound condition.

The logical operators and their meanings are:

<u>Logical Operator</u>	<u>Meaning</u>
OR	logical disjunction
AND	logical conjunction
NOT	logical negation

The syntax of a compound condition using the AND or OR operator is:

(condition)	{	AND	}	(condition)	[	{	AND	}	(condition)	]	...
		OR					OR				

The syntax of a compound condition using the NOT operator is:

NOT (condition)
-----------------

The results of the relationships between two conditions A and B are:

A	B	NOT (A)	(A) AND (B)	(A) OR (B)
true	true	false	true	true
true	false	false	false	true
false	true	true	false	true
false	false	true	false	false

Additional pairs of parentheses, enclosing subconditions, may be used to specify the order in which the compound conditions are to be evaluated.

The compound conditions are evaluated in the following order:

- 1 (first): Arithmetic expressions
- 2 : Relational operators / table operator
- 3 : Subconditions in parentheses
- 4 : Logical NOT operator
- 5 : Logical OR operator
- 6 (last): Logical AND operator

When a sequence of operators has the same order, the operators are executed in order of their occurrence from left to right.

Examples of compound conditions and their evaluation:

NOT (A > B)

First the relational condition  $A > B$  is evaluated, then the result is negated.

(X01 = X02) AND (A IN TAB1)

First the relational condition preceding the word AND is evaluated, then the table condition following AND, and finally the AND operator is executed.

NOT ( (X01 > A) OR (X01 < B) )

First the relational conditions preceding and following the word OR are evaluated, then the OR operator and finally the NOT operator.

NOT (X01 > A) OR (X01 < B)

First the relational condition following the word NOT is evaluated, and the result is negated, then the relational condition following the word OR is evaluated, and finally the OR operator is executed.

A statement is the basic unit of a field program or a subprogram. Each statement begins with a verb and describes some action to be taken. Normally this is an action which could not be specified in the checkbox part of the format coding sheet.

The statements are separated by commas, and a program is terminated by a comma.

A field program consists of none, one or more statements.

In the format language there are two categories of statements: conditional statements and unconditional statements. A conditional statement is one which contains some conditions that are tested to determine the path to be taken in the field program (the IF statement). An unconditional statement is one which specifies an unconditional action to be taken.

### 3.7.1 Unconditional Statements

3.7.1

#### 3.7.1.1 ALARM Statement

The ALARM statement is used to display error messages on the message part of the keystation screen. The syntax of the ALARM statement is:

ALARM	{ nonnumeric constant register register (subscripts) field name field name (subscripts) }
-------	---

The ALARM statement displays the contents of the operand on the second line (the message part) of the keystation screen. Statements following the ALARM statement are not executed, and the operator must either correct or bypass the field.

Examples of the ALARM statement:

```
ALARM 'FINAL PRICE NOT OK',  
CONNECT 'BATCH OUT OF BALANCE, DIFF=' TO X01 GIVING  
X02,  
ALARM X02,
```

### 3.7.1.2 ALLOW and DISALLOW Statements

The ALLOW and DISALLOW statements check current field for specific values. the ALLOW statement specifies allowable values, and the DISALLOW statement specifies incorrect values. The syntax for these statements is either:

{ ALLOW DISALLOW }	{ { + - } numeric constant nonnumeric constant }	{ { + - } numeric constant nonnumeric constant } ...
-----------------------------	--	--

or:

{ ALLOW DISALLOW }	[DISC] table name
-----------------------------	-------------------

In the second form of the ALLOW/DISALLOW statement table name is a name which identifies the table in the library of core tables (the DISC option is not used) or in the library of DISC tables (the DISC option is used). The table may be either single or multiple entered. Current field is checked against the table arguments.

The type of the constants following the word ALLOW or DISALLOW, or the type of table arguments must correspond with the type of current field. The methods for comparing numeric and nonnumeric operands are described in Sections 3.6.1.1 and 3.6.1.2.

In the ALLOW statement, if the contents of the current field are not one of the specified values, the statements following the ALLOW statement are not executed, and the operator must correct or bypass the field.

In the DISALLOW statement, if the contents of the current field are one of the specified values, the statements following the DISALLOW statement are not executed, and the operator must correct or bypass the field.

Examples of the ALLOW and DISALLOW statements:

```
ALLOW 'HANSEN' 'JENSEN',
ALLOW 0 125 512,
ALLOW CTABL,
```

where CTABL is the name of a CORE table

```
ALLOW 'A' 'ST' 'XYZ',
DISALLOW DISC TAB01,
```

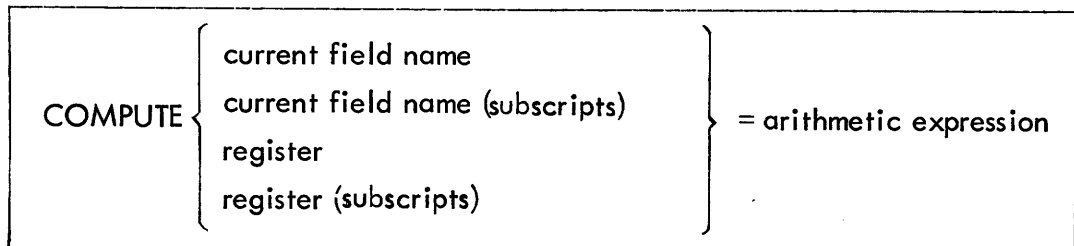
where TAB01 is the name of a DISC table

```
DISALLOW -2 -1 0 +1 +2,
```

### 3.7.1.3 COMPUTE Statement

3.7.1.3

The COMPUTE statement is used for arithmetic calculations. The syntax of the COMPUTE statement is:



The expression is evaluated (see Section 3.5) and the result is stored in the operand preceding the equal sign. The current field is allowed as destination only if it is a not keyed field and numeric in type.

Please note that nonnumeric operands containing only digits and fill characters may be entered in expressions by converting them to numeric types with the conversion operator NUMERIC.

Examples of the COMPUTE statement:

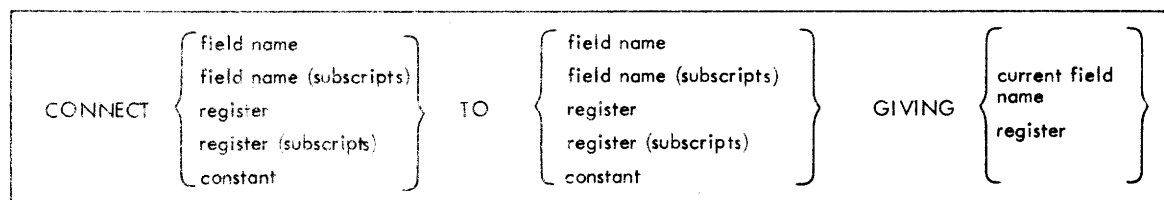
```
COMPUTE A1 = 0,
COMPUTE X01 = X01 + PRICE,
COMPUTE MONTH = DATE(3) * 10 + DATE(4),
COMPUTE X01(1) = F1,
COMPUTE X01(2) = X04(5),
COMPUTE FLD = FLD + 1,
COMPUTE X03 = NUMERIC X02 + 1,
COMPUTE X01 (6:7) = F1 (8:9),
```

Notice: The second example of the COMPUTE statement shows how you can make a total of a field (named PRICE) in a register. Each time PRICE is keyed the statement shown is executed and at the end of the registration it will hold the total.

### 3.7.1.4 CONNECT Statement

3.7.1.4

The CONNECT statement is used to connect two items and to store the resulting character string. The syntax of the CONNECT statement is:



The operands preceding and following the word TO are concatenated in left-to-right order. The resulting character string is stored in the operand following the word GIVING. Current field is only allowed as destination if it is a not keyed field and nonnumeric in type. The source operands can be either numeric or nonnumeric; if an operand is numeric it is interpreted as a nonnumeric character string.

Examples of CONNECT statements:

CONNECT A TO B GIVING X01,

where A='ABC' and B='DEF' causes X01='ABCDEF'.

CONNECT 'AMOUNT=' TO A1 GIVING X01,

where A1= 512 causes X01='AMOUNT= 512'.

CONNECT FLD TO '' GIVING X01,

where the lengths of FLD and X01 are equivalent, causes X01 = contents of FLD. If FLD is numeric in type this construction may be used to convert the contents of FLD from numeric to nonnumeric type.

CONNECT 'Δ' TO X01 GIVING X01,

This construction will shift the contents of X01 one position to the right and a space will be stored in the first position of X01.



### 3.7.1.5 DEFINE Statement

3.7.1.5

The DEFINE statement is used to define the length of a register in character positions. The syntax of the DEFINE statement is:

```
DEFINE register numeric constant
```

The numeric constant defines the register length in character positions. The upper limit for the register length is 255 character positions, but to conserve storage the register size should be as small as possible. The length of a register must be defined by a DEFINE statement before the register is used in any other statement, or before the register is used in connection with automatic duplication, insertion or incrementation (i.e., kind = D, C, or I). It is only allowed to redefine a register, if it is equivalent in length to the first definition.

Example of the DEFINE statement:

```
DEFINE X01 1,
```

Notice: The pseudo-registers mentioned in section 3.3.2 are pre-defined. The define statement is only used for defining registers named X01 - X99.

### 3.7.1.6 DISPLAY Statement

The DISPLAY statement is used to display operator information on the message part of the keystation screen. The syntax of the DISPLAY statement is:

```
DISPLAY { nonnumeric constant  
register  
register (subscripts)  
field name  
field name (subscripts) }
```

The DISPLAY statement displays the contents of the operand on the second line (message part) of the keystation screen, and it will be displayed until some other message to the second line occurs. The display statement can be used for simple fill-in-the-blanks keying and for debugging format programs by displaying register contents.

Examples of the DISPLAY statement:

```
DISPLAY 'KEY YOUR INITIALS ',  
DISPLAY X01,
```

3.7.1.7 DUP statement (Not yet available)

3.7.1.7

The DUP statement is used to simulate activation of the DUP control key.

The syntax of the DUP statement is:

```
DUP numeric constant FIELDS
```

The numeric constant defines the number of following Constant, Duplication, and Incrementation fields to be created as if the DUP control key was activated. If the number of fields reference to a field beyond the record a runtime error will occur as will be the result if a Keyed field is encountered before the specified number of fields have been treated. Otherwise the statements following the DUP statement are not executed and the action specified by the statement is performed.

Examples of DUP statement:

```
DUP 1 FIELD,  
IF FLD = X01 THEN DUP 3 FIELDS,
```

Notice: If an error, calling upon operator intervention, occurs before the number of fields have been treated then the action specified by the DUP statement is interrupted and a message will be displayed on the key station display screen as would be the case if the DUP key had been pressed the same number of times as specified by the statement.

3.7.1.8 END Statement

3.7.1.8

The END statement is used to terminate a format or a subprogram. It must physically be the last statement of the format or the subprogram. The syntax of the END statement is:

```
END
```

### 3.7.1.9 END SUBFORMAT Statement

3.7.1.9

The END SUBFORMAT statement is used to terminate any subformat except the last subformat, which is terminated by the END statement. When required, the END SUBFORMAT statement must physically be the last statement in the subformat. The syntax of the END SUBFORMAT statement is:

```
END SUBFORMAT
```

### 3.7.1.10 GOTO Statement

3.7.1.10

The GOTO statement is used to transfer control from one part of the program (that is, a field program or a subprogram) to another statement in the same program. The syntax of the GOTO statement is:

```
GOTO label
```

A statement may be labeled by assigning it a name followed by a colon (additional labels are allowed). The label is used in a GOTO statement to pass control to the statement after the label.

Labels must be defined within the program that contains the reference to the label, and a GOTO statement cannot reference a label in another program.

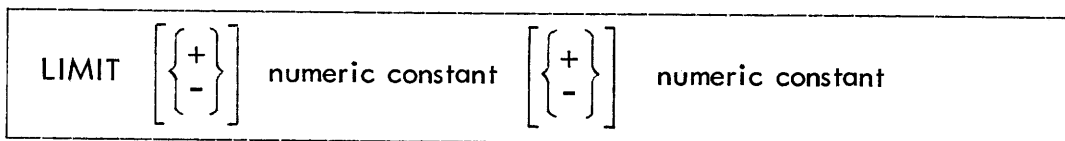
Examples of labels and the GOTO statement:

```
IF X01 < 0 THEN GOTO ERROR,  
-  
-  
ERROR: ALARM 'BATCH OUT OF BALANCE',  
  
AGAIN: IF X01 = 0 THEN GOTO NEXT,  
COMPUTE X01 = X01 - 1,  
-  
GOTO AGAIN,  
-  
NEXT:
```

### 3.7.1.11 LIMIT Statement

3.7.1.11

The LIMIT statement is used to check current field against a range of values. The syntax of the LIMIT statement is:



The LIMIT statement is only allowed if current field is numeric in type.

The second value must be greater than or equal to the first value. The range includes the smallest and the largest value.

If the check falls, the statements following the LIMIT statement are not executed, and the operator must either correct or bypass the field.

Examples of the LIMIT statement:

```

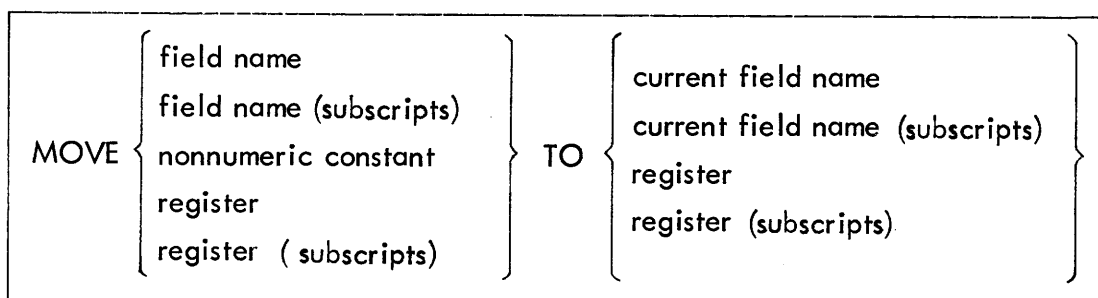
LIMIT 1000 5000,
LIMIT -500 999,
LIMIT -510 -509,
LIMIT -1 11,
LIMIT 0 0,

```

### 3.7.1.12 MOVE Statement

3.7.1.12

The MOVE statement is used for moving nonnumeric data from one place to another, such as from one field to another. The syntax of the MOVE statement is:



The operand preceding the word TO is moved to the operand following the word TO. Both operands must be nonnumeric in type. Current field name is only allowed as destination if it is a not keyed field.

Examples of the MOVE statement:

```
MOVE 'TEXT' TO X02,  
MOVE X01(1) TO FLD1(3),  
MOVE X03 TO FLD4,  
MOVE FLD(1) TO FLD(2),
```

### 3.7.1.13 NOTE Statement

3.7.1.13

The NOTE statement is used to write a commentary which is shown on the source listing but is not used in the system. The syntax of the NOTE statement is:

NOTE character string
-----------------------

The system ignores the character string following the word NOTE up to the first comma or semicolon.

Observe the following rules about the NOTE statement:

1. The character string may contain any characters including those not in the format language character set.
2. The character string may proceed through more than one line.
3. If quotation marks are used in the character string they must occur in pairs on a line.
4. The NOTE statement must not be the last statement before ELSE (because there is no comma or semicolon before ELSE, see Section 3.7.2.1 about the IF statement).
5. The character string may contain any reserved name.
6. The character string may be empty.

Examples of the NOTE statement:

```
NOTE CHECK DATE,  
NOTE THIS FIELD CONTAINS THE SALES PRICE,  
NOTE IF X01 <> 0 THEN FIELD 1 IS INCORRECT,  
NOTE 'JENSEN' IS AN INVALID NAME,  
NOTE 'JENSEN' IS AN INVALID NAME.  
    BUT 'HANSEN' IS OK;  
NOTE ,
```

Notice: A field program must not end with a label, because labels are preceding statements. If a label is wanted at the end of a field program, a NOTE statement with an empty character string may be used as last statement.

#### 3.7.1.14 PERFORM Statement

3.7.1.14

A PERFORM statement is used to pass control from a field program to a subprogram, or from one subprogram to another subprogram. The syntax of the PERFORM statement is:

PERFORM subprogram name
-------------------------

The subprogram name must be in the subprogram library (see Section 5.2). Return from the subprogram is made to the statement following the PERFORM statement.

The PERFORM statement may occur as the last statement in a field program, in which case further statements are not required.

See further in Section 3.8 about subprograms.

An example of a PERFORM statement is:

```
PERFORM CHE10,
```

#### 3.7.1.15 SEARCH Statement

3.7.1.15

The SEARCH statement is used to search a table for a specified argument and if the search is successful, to store the function (s) of the argument, otherwise if the search is unsuccessful, to perform a specific action. The syntax of the SEARCH statement is:

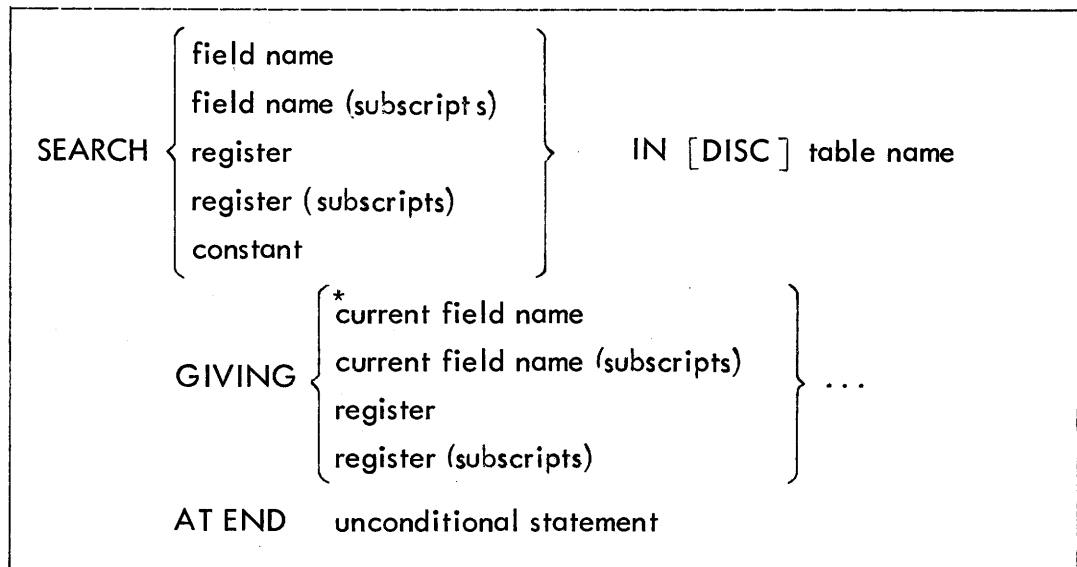


Table name identifies the table in the library of core tables (the DISC option is not used) or in the library of DISC tables (the DISC option is used). The table must not be single-entried.

Using the operand following the word SEARCH the table is searched for a match against the arguments of the table.

If the search is successful, the function(s) of the argument which matched will be stored in the operand(s) following the word GIVING, and the unconditional statement following AT END will not be executed.

Double-entried core-tables can only contain one function per argument, i.e. only one operand must be specified following the word GIVING.

If the table is a disc-table with more than one function per argument then as many operands as the number of functions per argument must be specified following the word GIVING. The first function corresponding to the argument will then be stored in the first operand specified, the second function in the second operand specified etc.

To prevent unwanted functions from being stored, an asterisk may be specified instead of one or more operands following the word GIVING. The place of an asterisk among the operand(s) defines the unwanted function(s).

If the search is unsuccessful, control is transferred to the unconditional statement following the words AT END, and the operand(s) after the word GIVING will not be changed.

The source operand must be of the same type as the table arguments. Current field is only allowed as destination if it is a not keyed field, and if it is of the same type as the table functions.

Examples of the SEARCH statement:

```
SEARCH CUSNO IN CTABL GIVING X01
AT END ALARM 'CUSTOMER NUMBER NOT KNOWN',

SEARCH FLD1 IN ATABL GIVING FLD4
AT END GOTO ERROR,

SEARCH FLD(1) IN ATABL GIVING X01
AT END COMPUTE X01 = 0,

SEARCH ARTNO IN DISC ARTTB
GIVING * X01 X13 **
AT END ALARM 'ARTICLE NUMBER NOT KNOWN',
NOTE ONLY 2ND AND 3RD ARGUMENT OUT OF 5 ARE STORED,

SEARCH CUSNO, IN CTABL GIVING *
AT END ALARM 'CUSTOMER NUMBER NOT KNOWN',
```

Notice: If the search is unsuccessful no value is stored in the destination operand (s).

### 3.7.1.16 SELECT Statement

3.7.1.16

The SELECT statement is used to change subformat under program control. The syntax of the SELECT statement is:

SELECT SUBFORMAT subformat name
---------------------------------

The SELECT statement may appear only in the last field program in a subformat. The statements following the SELECT statement are not executed, and the subformat change is made.

Examples of the SELECT statement:

```
SELECT SUBFORMAT 2,
IF X01 = 0 THEN SELECT SUBFORMAT E,
```



### 3.7.1.17 SET Statement

3.7.1.17

The SET statement is used to set the field status to valid or invalid. The syntax of the SET statement is:

SET field name { VALID INVALID }
-------------------------------------

The validity flag of the specified field is set valid or invalid, based upon the selected option.

Examples of the SET statement:

```
SET FLD1 VALID,  
IF TOTAL <> X01 THEN SET A INVALID,
```

Notice: The field status will not be replayed, i.e. once a field status is changed (in)valid it is treated as if it was (in)valid.

### 3.7.1.18 SKIP Statement

3.7.1.18

The SKIP statement is used to make an automatic skip to a forward field. The syntax of the SKIP statement is:

SKIP numeric constant FIELDS
------------------------------

The numeric constant defines the number of fields to be skipped, which must be greater than 0. If the number of fields reference to a field beyond the record a runtime error will occur, otherwise the statements following the SKIP statement are not executed and the skip action is performed. The skipped fields are filled with fill characters and their field programs are not executed.

Examples of the SKIP statement:

```
SKIP 1 FIELD,  
IF FLD > 10 THEN SKIP 2 FIELDS,
```

## 3.7.2 Conditional Statements

3.7.2

### 3.7.2.1 IF Statement

3.7.2.1

The IF statement is a conditional statement. It is used to make a path through the field program, depending on the result of the evaluation of the specified

condition. The syntax of the IF statement is:

IF condition THEN sentence [ELSE sentence]

The condition following the word IF is evaluated. If the condition is true, the sentence following the word THEN is executed. Control is then passed to the next statement after the IF statement, unless the sentence contains a GOTO statement, in which case control is passed to the GOTO label.

If the condition is false, the sentence following THEN is skipped and the sentence following the word ELSE is executed, or, if the ELSE option is omitted, the next statement after the IF statement is executed.

The IF statement may occur as last statement in a field program, in which case no following statement is required.

A sentence contains one or more statements, separated by a semicolon, and terminated with a comma or the word ELSE.

A sentence following the word THEN may contain any statement except a conditional statement. A sentence following the word ELSE may contain any statement including a conditional statement.

Notice:

1. Neither comma nor semicolon is allowed immediately before the word ELSE.
2. A sentence following the word THEN is terminated when the word ELSE or a comma is encountered.
3. A sentence following the word ELSE is terminated when the first comma is encountered.

Examples of the IF statement:

```
IF NOT (TOTAL = X04) THEN 'ALARM TOTAL PRICE NOT OK'  
ELSE DISPLAY 'END OK',
```

```
IF X01 > 0 THEN COMPUTE X02 = X02 + A;  
COMPUTE X01 = X01 - 1,
```

IF X01 < X02 THEN COMPUTE X02 = X02 - A  
 ELSE COMPUTE X02 = X02 + A; COMPUTE X03 = X03 - 1,  
 IF A > B THEN GOTO C,

NOTE NEXT STATEMENT CHECKS IF A PREVIOUS (NUMERIC)  
 FIELD ONLY CONSISTS OF FILL CHARACTERS (I.E. HAS BEEN  
 SKIPPED), IF ALPHANUMERIC FLD5 = '\*\*\*\*\*' THEN SKIP 2 FIELDS,

## 3.8 Subprograms

3.8

A subprogram is a program that is called from another program, by using the PERFORM statement. Control is transferred to the first statement in the subprogram, and returned to the calling program by the END statement in the subprogram.

### 3.8.1 Statements In Subprograms

3.8.1

The statements allowed in subprograms are:

ALARM	(field name not allowed as operand)
ALLOW	(the DISC table option is not allowed)
COMPUTE	(field names not allowed as operands)
CONNECT	(field names not allowed as operands)
DEFINE	
DISALLOW	(the DISC table option is not allowed)
DISPLAY	(field name not allowed as operand)
DUP	
END	
GOTO	
IF	(validity condition and field names as operands are not allowed)
LIMIT	
MOVE	(field names not allowed as operands)
NOTE	
PERFORM	
SEARCH	(the DISC table option is not allowed)
SKIP	

Notice:

1. A subprogram must end with an END statement.
2. A subprogram cannot call itself in a PERFORM statement.

3.8.2 Operands In Subprograms

3.8.2

Only registers and constants are allowed as operands in statements in subprograms. The register names refer to the same registers as in the field programs, and a register may be used both in field programs and in subprograms.

Registers are used for transferring data to and from subprograms, as the format language contains no possibilities of defining subprograms with parameters.

Example:

Consider a problem, where you want a check-digit control in a numeric field. You program the check in a subprogram, for example:

SUB.PROG NAME	COMMENT
1	2
C0001CHECK-DIGIT CONTROL - X01=CUSTOMER NUMBER	
PROGRAM STATEMENTS	
1	
IF (X01(1)*5	
+X01(2)*4	
+X01(3)*3	
+X01(4)*2	
+X01(5)*1) MOD 11 <> 0 THEN	
ALARM 'INCORRECT CUSTOMER NUMBER',	
END,	

The corresponding field program may look as follows:

FORMAT NAME	S	P	COMMENT
1	2	3	4

FIELD NAME	PKG	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	RL	FILL	REFL	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CUSNO				5		ON	1							COMPUTE X01=CUSNO, PERFORM C0001,



## 4 Execution of Format Programs

### 4.1 Selecting Subformat 4.1

After invoking a format the system is set ready to execute the first subformat.

The same subformat is run through cyclically until a new subformat is selected.

A subformat can be selected manually by using the SUBFORMAT control key, or it can be program-selected by execution of a SELECT SUBFORMAT statement.

### 4.2 Terminating a Format Program 4.2

A format program is terminated by executing an END statement.

### 4.3 Execution of Subformats 4.3

The field descriptions in a subformat are executed sequentially. After the initial sequential execution of all field descriptions in a subformat follows a cyclical repetition starting with the subformat's first field description, and so on.

This sequential process can be interrupted by a SKIP statement, or by using the RECORD RELEASE or the FIELD BACK control key.

By using the SKIP statement the execution of one or several subsequent field descriptions in a subformat can be skipped.

Pressing the RECORD RELEASE control key will cause the remaining field descriptions of a subformat to be skipped, provided this is allowed - see Section 4.4.6.

Pressing the FIELD BACK control key will cause a backward step in the field sequence of a subformat - though not beyond the current record.

## 4.4 Execution of a Field Description

4.4

The execution of a field description is dependent on the field definition, the field input and previous skip instructions. There is a number of possible alternatives:

- \* The field is a KEYED field.
- \* The field is an AUTOMATIC field.
- \* The field is a NOT KEYED field (including 0-length fields).
- \* The field has been skipped by a SKIP statement.
- \* The field has been skipped by the ENTER key, i.e., no data input.
- \* The field has been skipped by the RECORD RELEASE key.
- \* The field has been skipped by the BYPASS key.

### 4.4.1 Keyed Fields

4.4.1

If a field is keyed (i.e. kind = K), the following steps will be executed in the named order:

- \* Right/left justification and insertion of fill characters.
- \* Length check.
- \* Minimum length check.
- \* Type check.
- \* Execution of the field program.

### 4.4.2 Automatic Fields

4.4.2

There are three kinds of automatic fields:

DUPLICATE fields (i.e. kind = D),  
CONSTANT fields (i.e. kind = C),  
INCREMENT fields (i.e. kind = I).

#### 4.4.2.1 Duplicate Fields. Two possibilities are open:

4.4.2.1

1. Keying the field;
2. Duplicating the field.

Keyed fields are executed as specified in 4.4.1.



Subsequently,

- \* the field's contents are transferred to the register defined in the field definition. Un equal length of field and register cause a runtime error.

Pressing the DUPLICATE control key will cause the following to happen:

- \* The contents of the register specified in the field description are transferred to the field. Un equal length of field and register cause a runtime error.
- \* A typecheck is performed.
- \* Execution of the field program.
- \* The contents of the field are transferred to the corresponding register.

4.4.2.2 Constant Fields. There are two possibilities:

4.4.2.2

1. Keying the field;
2. Duplicating the field.

Keyed fields are executed as specified in 4.4.1.

Pressing the DUPLICATE control key will result in the field being executed as a DUPLICATE field, cf. 4.4.2.1. The contents of the field will not be transferred to the register after the execution of the field program.

4.4.2.3 Increment Fields. There are two possibilities:

4.4.2.3

1. Keying the field;
2. Duplicating the field.

Keyed fields are executed as specified in 4.4.1.

Subsequently,

- \* the field's contents are transferred to the register defined in the field definition. Unequal length of field and register cause a runtime error.

Pressing the DUPLICATE control key will have the following result:

- \* The contents of the register specified in the field definition are transferred to the field. Unequal length of field and register cause a runtime error.
- \* The field is incremented by 1.

- \* A typecheck is performed.
- \* Execution of the field program.
- \* The contents of the field are transferred to the corresponding register.

4.4.3     Not Keyed Fields     4.4.3

NOT KEYED fields are fields with kind = N or length = 0. Such fields will have no input during registration.

The following is performed:

- \* Insertion of fill characters.
- \* Execution of the field program.

4.4.4     Fields Skipped by SKIP     4.4.4

If a field is skipped by using the SKIP statement, the following will occur:

- \* Insertion of fill characters.
- \* No execution of the field program.
- \* Registers corresponding to duplicate or increment fields will not be updated.

4.4.5     Fields Skipped by ENTER     4.4.5

If a field is bypassed by simply pressing the ENTER key, the following will occur:

- \* Minimum length check, i.e. if minimum length is greater than zero, an error message appears and the field has to be keyed.

Otherwise, the following is performed:

- \* Insertion of fill characters.
- \* No execution of the field program.
- \* Registers corresponding to duplicate or increment fields will not be updated.

4.4.6     Fields Skipped by RECORD RELEASE     4.4.6

Fields skipped by pressing the RECORD RELEASE key will be executed in accordance with the field's definition.

4.4.6.1     Fields with Kind KEYED, DUPLICATE, CONSTANT, INCREMENT. Fields with kind KEYED, DUPLICATE, CONSTANT, INCREMENT are executed as fields skipped with ENTER (see Section 4.4.5).

- \* If the minimum length of the field is not zero, an error message appears, and the field must be keyed. Subsequently, RECORD RELEASE is stopped, and normal execution is resumed in KEY, REKEY, or EDIT mode.

4.4.6.2 Fields with Kind NOT KEYED. If the skipped field is of the NOT KEYED kind the following will happen:

- \* Insertion of fill characters.
- \* Execution of the field program.

4.4.7 Fields Skipped by BYPASS 4.4.7

The BYPASS control key is used to bypass fields that one has given up keying correctly.

- \* The field will retain the contents it had before activating the BYPASS key, if the field has been keyed.
- \* The field program is not executed.
- \* Registers corresponding to duplicate and increment fields are not updated.

4.4.8 Execution of a Field Program 4.4.8

The statements in a field program are executed sequentially.

The whole field program is executed. However, any error detected through ALLOW, DISALLOW, or LIMIT, as well as execution of an ALARM, DUP, SKIP, or SELECT statement will cause the field program to be interrupted after such a statement.

The sequential processing of a field program can be interrupted by a GOTO statement.

4.5 Field Flags 4.5

During execution of a field description the field is assigned two flags, which are independent of each other. They are:

- Validity flag
- Skipped flag.

#### 4.5.1 Validity Flag

4.5.1

This flag has two values:

VALID

INVALID

An INVALID flag is assigned to a field which

- \* is skipped by the BYPASS key, or
- \* is set INVALID by a SET statement, or
- \* is a 'NOT KEYED' field containing an error, or
- \* is a keyed field containing an error, that has not yet been corrected.

In other cases the field gets a VALID flag.

If a field in a record has an INVALID flag, the record will also get an INVALID flag.

You may ask for a field's validity flag in an IF statement.

#### 4.5.2 Skipped Flag

4.5.2

This flag has three values:

NOT SKIPPED

SKIPPED

SKIPPED BY STATEMENT

A SKIPPED flag is given to a field which

- \* is skipped by the ENTER key, or
- \* is skipped by the BYPASS key, or
- \* is skipped by the RECORD RELEASE key.

A SKIPPED BY STATEMENT flag is given to a field which

- \* is skipped by the SKIP statement.

Otherwise, the field gets a NOT SKIPPED flag. We say the field is skipped if it has a SKIPPED flag or a SKIPPED BY STATEMENT flag and the field program is not executed.

### 4.5.3 Flags for REKEY

4.5.3

For fields specified as 'REKEY YES' the flags in REKEY mode are set as for KEY mode when the fields are rekeyed. A field specified as 'REKEY NO' gets normally the same flags as the corresponding old field. But depending on the old field flags for 'REKEY NO' fields some special actions occur:

If the old field validity flag is INVALID, the old field is not used as field input but an error message appears and the field must be keyed as if it were specified as 'REKEY YES'.

If the old field skipped flag is SKIPPED BY STATEMENT and the field will not again be skipped by the SKIP statement, an error message appears and the field must be keyed as if it were specified as 'REKEY YES'.

### 4.5.4 Flags for EDIT

4.5.4

When fields are keyed in EDIT mode the flags are set as for KEY mode. When searching in EDIT mode the fields are normally given the same flags as the corresponding old fields. But depending on the old field flags some special actions occur:

If the old field validity flag is INVALID, the old field is not used as input, but the searching stops with an error message.

If the old field skipped flag is SKIPPED BY STATEMENT and the field will not again be skipped by the SKIP statement, the old field is not used as input, but the searching is stopped with an error message.

## 4.6 Registers

4.6

Registers may be assigned directly in the field program, or indirectly when used for automatic fields (kind: duplicate or increment). In the latter case the field is not transferred to the register until after execution of the field program, so that the contents of the register will be exactly the same as the contents of the field.

When the format program is executed during keying, the format is said to be 'playing' the batch. The format execution sequence and the contents of the registers may be dependent on what is keyed. Therefore, a register of which the contents are affected by what is keyed must be changed whenever the field affecting it is changed. To revise the register contents and bring them up to date, the entire batch must be 'replayed' from start up to the point where the change is made. To save time during replay pictures of the registers are frequently saved in the batch, and the replay is actually performed from the nearest preceding register picture.

Replay occurs:

- \* when the RECORD BACK key is used.
- \* when the FIELD BACK key is used.
- \* when the CLEAR key is used.
- \* when the SUBFORM key is used.
- \* when the RECORD key is used.
- \* when an error is detected during format program execution and a register has been changed in the format program.
- \* when keying or rekeying is reopened after the ESCAPE key has been used.

Be aware that the status of fields will not be replayed.

The execution of the format image belonging to a given format is given by the execution of the format itself.

In selecting subformat the corresponding subformat image is automatically used.

If a subformat image consists of only one page, the fill-in-the-blanks mask of this page is written on the screen on selection of the subformat, and remains there during the repeated registration of this subformat.

If a subformat image consists of several pages, the corresponding fill-in-the-blanks mask will be replaced during the run of the subformat. This replacement occurs as each new page is specified in the field definitions.

## 5 Entering New Formats, Subprograms, and Tables

The keying of new formats, subprograms, or tables is controlled by a standard format. The result is a batch, which can be edited, saved, listed, etc., in the same way as with all other batches in the Data Entry system.

Before entering new formats or subprograms in the system all referenced subprograms and tables must have been entered.

The entering of formats, subprograms, and tables is guided by the supervisor program TRANSLATE, with the exception of DISC tables which are treated separately.

TRANSLATE controls if the source batch is correctly structured and, if so, translates it into internal form. The name of the translated file is then included in the current library, after which the new format, or subformat, or table, can be referenced by its name.

### 5.1 New Formats

5.1

The format text of the coding sheets is keyed under the control of the standard format FORM.

The keying of tags written on complementary coding sheets is controlled by the standard format IMAGE.

Both standard formats support fill-in-the-blanks guidance.

The FORM standard format contains 3 subformats (see Appendix IV):

- H - reads subformat head belonging to the new format;
- F - reads field description belonging to the new format;
- E - terminates the batch (format text).

The format coding sheets are interpreted by FORM in the following way:

FORMAT NAME	S	P	COMMENT
1	2	3	4

1st subformat is H

- read by H

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	REKEY	DISPLAY KIND	REGISTER	PROGRAM STATEMENTS	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

automatic selection of F

- read by F  
- read by F

END SUBFORMAT,\*

- read by F

automatic selection of H

FORMAT NAME	S	P	COMMENT
1	2	3	4

- read by H

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	REKEY	DISPLAY KIND	REGISTER	PROGRAM STATEMENTS	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

automatic selection of F

- read by F  
- read by F

END,\*

- read by F

automatic-ally terminated by E

\* Such automatic selection of subformats is performed only if this statement is indicated exactly as above, meaning that the statement must begin in the 1st position in column 15, that there must be one, and only one, space between END and SUBFORMAT, and that the words SUBFORMAT and END must be immediately followed by a comma.

Subformat H contains 5 field descriptions, which check the following:

1. FORMAT NAME (AN)

The first character of the FORMAT NAME must be a letter from 'A' - 'Z'.

2. SUBFORMAT NAME (AN)

must be a letter from 'A' - 'Z' or a digit.



- |   |  |
|---|--|
| 3. PROTECTED (A)                          | 'Y', 'N', or 'Δ'                               |
| 4. COMMENT (AN)                           | No check.                                      |
| 5. Field description<br>(not keyed field) | No check; automatic selection of sub-format F. |

Subformat F contains 16 field descriptions, with the first 15 receiving input from the columns of the format coding sheets, while the 16th is a not keyed (0-length) field for consistency checks of the first 15 field descriptions. The following checks are performed.

- |                   |  |
|-------------------|--|
| 1. FIELDNAME (AN) | The first character of FIELDNAME must be a letter from 'A' - 'Z', or FIELDNAME = 'ΔΔΔΔΔ'.  |
| 2. PAGE (N)       | When PAGE is keyed as SPACE ENTER two field descriptions are automatically skipped (that is, neither PAGE nor LINE or POSITION are specified); else the following checks are made: PAGE must be a number from 1 to 8, and must, furthermore, be greater than or equal to the last defined PAGE of the current subformat. |
| 3. LINE (N)       | LINE is a number from 1 to 21.   |
| 4. POSITION (N)   | POSITION is a number from 1 to 80.   |
| 5. LENGTH (N)     | When LENGTH is defined as 0 (zero) or as SPACE ENTER, an automatic skipping occurs to the 15th field description (PROGRAM STATEMENTS); else the following check is made: LENGTH must be a number from 1 to 80.   |
| 6. MIN.LENGTH (N) | MIN.LENGTH must be less than or equal to LENGTH, and has to be a number from 0 to 80.  |
| 7. TYPE (A)       | 'NΔ', 'SN', 'SS', 'AN', or 'AΔ'.   |

8. OUTPUT POSITION (N)	OUTPUT POSITION is a number from 0 to 255.
9. R/L (A)	'L', 'R', or 'Δ'
10. FILL (AN)	'Δ', '0', or '*'
11. REKEY (A)	'Y', 'N', or 'Δ'
12. DISPLAY (A)	'N', 'Y', or 'Δ'
13. KIND (A)	'Δ', 'D', 'C', 'I', 'N' or 'K'; furthermore, TYPE must be NΔ, if KIND = 'I'.
14. REGISTER (N)	REGISTER is a number from 1 to 99.
15. PROGRAM STATEMENTS (AN)	No checks.
16. Field description (not keyed field)	<p>IF LENGTH = 'ΔΔ', or LENGTH = 0: only PROGRAM STATEMENTS may be specified.</p> <p>The following applies if LENGTH &gt; 0:</p> <ul style="list-style-type: none"> <li>- Either PAGE, LINE, and POSITION are all specified, or none of them.</li> <li>- MIN.LENGTH, TYPE, and OUTPUT POSITION are specified.</li> <li>- If REGISTER is <u>not</u> specified, then KIND is either 'N', 'K', or 'Δ'.</li> <li>- If REGISTER is specified, then KIND is either 'C', 'D', or 'I'.</li> </ul> <p>When PROGRAM STATEMENTS start with 'END SUBFORMAT,', subformat H is automatically selected.</p> <p>When PROGRAM STATEMENTS start with 'END,', subformat E is automatically selected.</p>

Subformat E consists of one field description, with a field definition describing a not keyed (0-length) field, and the field program is solely an END statement.

In addition to normal error messages (length, type, limit, etc.), the following alarm texts may appear when using the standard format FORM:

	From subformat. Column
CURRENT PAGENO LESS THAN PREVIOUS PAGENO	F.2
ERROR IN CHECKBOX CONTENTS	F.16
ILLEGAL FIELD NAME	F.1
ILLEGAL FORMAT NAME	H.1
ILLEGAL SUBFORMAT NAME	H.2
KIND "I" ONLY ALLOWED IF TYPE = "N"	F.13
MIN.LENGTH GREATER THAN FIELD LENGTH	F.6

The standard format IMAGE contains 3 subformats (see Appendix IV).

H - reads subformat head belonging to the new format image;

F - reads tag description;

E - terminates batch (format image text).

The format image coding sheets are interpreted by IMAGE in the following way:

FORMAT NAME		S	COMMENT
1		2	3

PAGE	LINE	POSITION	TEXT
1	2	3	4

1st subformat is H

- read by H

automatic selection of F

- read by F  
- read by F

- read by F

manual selection of H

- read by H

automatic selection of F

- read by F  
- read by F

- read by F

manual selection of E terminates the batch

Subformat H contains 4 field descriptions, where the following checks are made:

- |   |  |
|---|--|
| 1. FORMATNAME (AN)                        | The first character of the FORMAT-NAME must be a letter from 'A' to 'Z'. |
| 2. SUBFORMAT NAME (AN)                    | is a letter from 'A' to 'Z', or a digit.                                 |
| 3. COMMENT (AN)                           | No check.  |
| 4. Field description<br>(not keyed field) | No check; automatic selection of subformat F.                            |

Subformat F contains 4 field descriptions, where the following checks are made:

- |                 |   |
|-----------------|---|
| 1. PAGE (N)     | PAGE must be a number from 1 to 8, and must, furthermore, be greater than or equal to the last defined PAGE of the current subformat. |
| 2. LINE (N)     | LINE is a number from 1 to 21.  |
| 3. POSITION (N) | POSITION is a number from 1 to 80.  |
| 4. TEXT (AN)    | No check.   |

Subformat E consists of one field description, with a field definition describing a not keyed (0-length) field, and the field program is solely an END statement.

In addition to normal error messages (length, type, limit, etc.), the following alarm texts may appear when using the standard format IMAGE:

	From subformat. Column
CURRENT PAGENO LESS THAN PREVIOUS PAGENO	F. 1
ILLEGAL FORMAT NAME	H. 1
ILLEGAL SUBFORMAT NAME	H. 2

When the created batches are considered to be correct, translation of the format is initiated by activating the supervisor program TRANSLATE (see Users Guide):

```
TRANSLATE FORM form-batch [image-batch]
```

Remember that all referenced subprograms and tables must be known to the system before starting the format translation. Once the format is translated they are no longer needed.

When the format has been correctly translated, the format name is included in the format library.

The subprogram text of the coding sheets is keyed under the control of the standard format SUBPR. The standard format supports fill-in-the-blanks guidance.

The SUBPR standard format contains 3 subformats (see Appendix IV):

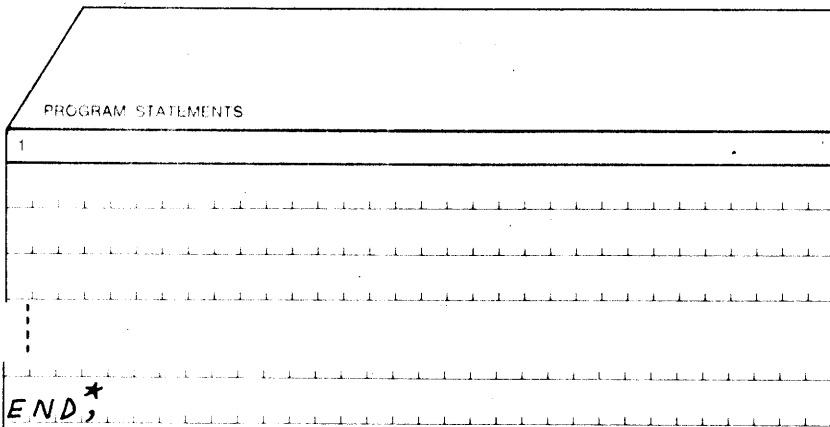
- H - reads subprogram head;
- P - reads a part of a subprogram;
- E - terminates the batch (subprogram text).

The subprogram coding sheets are interpreted by SUBPR in the following way:

1st subformat is H

SUB PRG NAME	COMMENT
1	2

- read by H



automatic selection of P

- read by P  
- read by P

- read by P

automatically terminated by E.

\* Automatic selection of subformats is accomplished only if this statement is indicated exactly as above. That is, the statement should start in the 1st position, and the word END should be immediately followed by a comma.

Subformat H contains 3 field descriptions, with the following checks to be made:

1. SUBPROGRAMNAME (AN)            The first character of the SUBPROGRAM-NAME must be a letter from 'A' to 'Z'.
2. COMMENT (AN)                    No check.
3. Field description  
    (not keyed field)                No check; subformat P is automatically selected.

Subformat P consists of one field description, with the following operation:

1. PROGRAM STATEMENTS (AN)      No check; subformat E is automatically selected if PROGRAM STATEMENTS start with 'END,'.

Subformat E consists of one field description, with a field definition describing a not keyed (0-length) field, and the field program being solely an END statement.

In addition to normal error messages (length, type, limit, etc.) the following alarm text may appear when using the standard format SUBPR:

	From subformat. Column
ILLEGAL SUBPROGRAM NAME	H.1

When the created batch is considered to be correct, translation of the subprogram is initiated by activating the supervisor program TRANSLATE (see Users Guide)

TRANSLATE SUBPROGRAM subpr - batch

Remember that all referenced subprograms and tables must be known to the system before starting translation of the subprogram. Once the translation is finished they are no longer needed.

When the subprogram has been correctly translated, it is included in the subprogram library.



## 5.3 New Tables 5.3

### 5.3.1 New Core Tables 5.3.1

The keying of the table text from the coding sheets is controlled by the standard format TABLE. The standard format supports fill-in-the-blanks guidance.

The TABLE standard format contains 15 subformats (see Appendix IV):

- H - reads columns 1 - 6 of table head.
- M - reads column 7 of table head.
- T - reads columns 8 - 14 of table head.
- 1 - reads table element, Single entry table, A-type = N;
- 2 - reads table element, Single entry table, A-type = AN;
- 3 - reads table elements, Double entry table, A-type = N, F-type = N;
- 4 - reads table elements, Double entry table, A-type = N, F-type = AN;
- 5 - reads table elements, Double entry table, A-type = AN, F-type = N;
- 6 - reads table elements, Double entry table, A-type = AN, F-type = AN;
- C - stores a control word (DISC TABLE) in the batch and selects subformat O;
- O - reads operation and selects subformat S;
- S - selects subformat A, or N, or O;
- A - reads disc table element of type AN and selects subformat S;
- N - reads disc table element of type N and selects subformat S;
- E - terminates batch (table text).

Subformats M, T, C, O, S, A, N, and E which concern disc tables (possibly Multiple entry tables) are explained in section 5.3.2.

The table coding sheets are interpreted by TABLE in one of the following two ways (see Appendix IV) when the table is a core-table:

1. Single entry table

1st subformat  
is H

TABLE NAME	T	A TYPE	A LGTH	F TYPE	F LGTH
1	2	3	4	5	6
	S				

- read by H

1	ARGUMENTS

automatic se-  
lection of sub-  
format 1 or 2

- read by 1 or 2

- read by 1 or 2

- read by 1 or 2

Manual selec-  
tion of E ter-  
minates batch.

## 2. Double entry table

1st subformat  
is H

TABLE NAME	T	A TYPE	A LGTH	F TYPE	F LGTH
1	2	3	4	5	6
	D				

- read by H

automatic se-  
lection of sub-  
format 3, 4,  
5, or 6

1 X 2 ARGUMENTS & FUNCTIONS	
1	
2	
1	
2	
1	

- read by 3, 4,  
5, or 6

- read by 3, 4,  
5, or 6

2	
1	
2	

- read by 3, 4,  
5, or 6

Manual selec-  
tion of E ter-  
minates batch.

Subformat H contains 7 field descriptions, with the following checks to be made:

1. TABLENAME (AN)
 

The first character in TABLENAME must be a letter from 'A' to 'Z'.  
'S' or 'D' or 'M'  
If TYPE = 'M' the next four field descriptions are skipped.
2. TYPE (A)
 

'NΔ' or 'AN'
3. ARGUMENTTYPE (A)  
(A-TYPE)
4. ARGUMENTLENGTH (N)  
(A-LGTH)
 

ARGUMENTLENGTH must be a number from 1 to 80.  
If TYPE = 'S' the next two field descriptions are skipped.

- |   |  |
|---|--|
| 5. FUNCTIONTYPE (A)<br>(F-TYPE)           | 'NΔ' or 'AN'   |
| 6. FUNCTIONLENGTH (N)<br>(F-LGTH)         | FUNCTIONLENGTH must be a number from 1 to 80.  |
| 7. Field description<br>(not keyed field) | No check; depending on TYPE, ARGUMENT-TYPE and FUNCTIONTYPE a subformat from 1 through 6 or M is automatically selected. |

Subformats 1 and 2 contain one field description each, with field type being N and AN, respectively - otherwise no check performed.

Subformats 3, 4, 5, and 6 contain 2 field descriptions each, with field types being N/N, N/AN, AN/N, and AN/AN, respectively - otherwise no check is performed.

Subformat E consists of one field description, with a field definition describing a not keyed (0-length) field, and the field program is solely an END statement.

In addition to normal error messages (length, type, limit, etc.) the following alarm text might appear when using the standard format TABLE:

	From subformat. Column
ILLEGAL TABLE NAME	H.1

When the created batch is considered to be correct, translation of the table is initiated by activating the supervisor program TRANSLATE (See Users Guide)

TRANSLATE TABLE table-batch

When the table has been correctly translated, the table name is included in the table library.

When using a table with many table elements, it is sometimes necessary to set up the table as a disc table. A disc table also features the possibility of up to seven table columns (one argument - six functions) whereas a core table can consist of at most two columns.

The structure of the table (number of columns, type and length of each column, largest number of entries) is set up with the help of supervisor program CREATE which creates a table consisting of empty entries and which includes the table name in the disc table library.

The entering of data to the table is accomplished with the help of supervisor program DISCTABLE. This program checks if the table text is correctly structured, and translates the data into table entries. The translation, during which the entries are stored in a hash-organized way, is usually a time-consuming process, but it will, on the other hand, enable quick checks to be made on the existence of certain elements, even in cases involving very large tables.

Please note, that it is possible to insert, update (replace), or to delete, elements in the translated table also with the help of supervisor program DISCTABLE, so as to avoid repeating the entire translation procedure.

The table text is either keyed to a batch under the control of the standard format TABLE, or it may be stored on a magnetic tape generated by another computer.

The subformats of format TABLE are listed in section 5.3.1, the interpretation of the coding sheets takes place in the following way:

TABLE NAME	T	A--	A	T	S
		TYPE	LOGN	TYPE	LGTH
1	2	3	4	5	6
	M				

NO. OF FS
7

A--	F1--	F2--	F3--	F4--	F5--	F6--
TYPE	TYPE	TYPE	TYPE	TYPE	TYPE	TYPE
ONLY TO BE KEYED IF FILLED IN						
8	9	10	11	12	13	14

1st subformat is H

- read by H

automatic selection of subformat M

- read by M

automatic selection of subformat T

- read by T. The subformat is automatically reselected up to six times if required (number of functions)

automatic selection of subformat O

- read by O

automatic selection of subformat A or N

ONLY TO BE KEYED IF FILLED IN, OP, F, O, D, E.	
OP	ARGUMENTS & FUNCTIONS
1	2
...	...
E	

- read by A or N

automatic selection of subformat A or N if following lines contain function(s)

- read by A or N

subformat A or N is automatically reselected as long as functions follow where upon subformat O is selected

- read by O

- read by O

automatic selection of subformat E.

Subformat H and E are explained in section 5.3.1

Subformat M contains three field descriptions with the following checks to be made:

1. Field description  
(not keyed field) No check, used for register definition.
2. NO. OF FUNCTIONS (N)  
(NO. OF F'S) NO. OF FUNCTIONS must be a number from 0 to 6.
3. Field description  
(not keyed field) No check, used for computations (e.g. the number of times to execute subformat T); automatic selection of subformat T.

Subformat T contains two field descriptions.

1. TYPE (A)  
(A-TYPE; F1,2,3,4,5,6-TYPE) 'NΔ' or 'AN'
2. Field description  
(not keyed field) No check, used for determining if keying is to continue under the control of this subformat or if subformat C is to be selected (depends on NO. OF FUNCTIONS).

Subformat C consists of one not-keyed field which stores the text 'DISCTABLE' in the batch so that compatibility with a magnetic tape (see section 5.3.2) is achieved.

Subformat O contains two fields:

1. OPERATION (A)  
(OP) 'I' or 'U' or 'D' or 'E'
2. Field description  
(not keyed field) No check, not keyed (0-length) field selecting subformat E if OPERATION = 'E' or, if not, selecting subformat S.

Subformat S consists of one not keyed (0-length) field which selects subformat A or N if entered from O. If entered from A or N then one of these subformats are reselected if a function is to be read, otherwise subformat O is selected.

Subformats A and N contain one field description each, with field type being AN and N, respectively - otherwise no check is performed.

If the table text is stored on magnetic tape, this must have the following format:

The record format of the file must be variable, blocked type (IBM VB format, max. 512 bytes), and the coding of characters must be in accordance with the ASCII or EBCDIC alphabets.

The first two records are used for identification:

record 1, 5 bytes : table name, fill characters: space.  
record 2, 9 bytes : the text DISC TABLE.

The following records concern the entries of the table.

An insertion of a table entry is accomplished by a sequence of 2 - 8 records:

record 1, 1 byte : the text I  
record 2, max. 80 bytes: argument of entry  
record 3, max. 80 bytes: function 1  
:  
record 8, max. 80 bytes: function 6

If the table is single-entried then an insertion consists of only two records. If the table is multiple entried, the number of records following record 2 corresponds to the number of functions per table entry.

A table entry is updated (replaced) by supplying the following sequence of 2 - 8 records:

record 1, 1 byte : the text U  
record 2, max. 80 bytes: argument of entry  
record 3, max. 80 bytes: function 1  
:  
record 8, max. 80 bytes: function 6

The number of necessary records is as for insertion as all elements of the entry are replaced.



A table entry is deleted from the table by a sequence of 2 records:

record 1, 1 byte : the text D

record 2, max. 80 bytes : argument of the entry

The last record in the file must be a one-byte record containing the text E.

The arguments and functions must conform the structure of the table with respect to type and length, see also section 2.3.3.

The table text translation is performed by activating the supervisor program DISCTABLE through one of the calls below (see Users Guide):

DISCTABLE BATCH.batchname (if the table text is keyed  
under control of the TABLE format)

DISCTABLE MTx.filename ASCII (if the table text is stored on mag-  
netic tape in ASCII code; x = tape  
station no.)

DISCTABLE MTx.filename EBCDIC (if the table text is stored in  
EBCDIC code).



## 6 Programming Hints

This section describes the special facilities of the system, how they are programmed, and how they work when the finished format is used for keying.

### 6.1 Screen Processing 6.1

Screen processing is understood as covering all information given by the format program on the utilization of the screen's data area. The first lines on the screen are always reserved for the system.

As an aid to registration it is possible to specify tags, and it is likewise possible to specify the keying position of the single units on the screen. Screen processing, can, however, also be entirely left to the system.

#### 6.1.1 Screen Processing Assigned To the System 6.1.1

In case one chooses to leave screen processing entirely to the system, the columns PAGE, LINE, and POSITION in the definition sections of all field descriptions are left unkeyed.

The system will then utilize the screen in such a way that LENGTH in the definition sections of the field descriptions assigns the number of screen positions that are set aside for keying to the field.

- The first field of the subformat is keyed in the left most position of the first data line on the screen. This field is defined as the first field on the subformat's first page.
- If there is sufficient space on the current line, one proceeds to key the next field on this line, leaving one blank position before the field.
- If there is not enough space left on the current line, the next field is keyed on the following line, starting with the left most position.

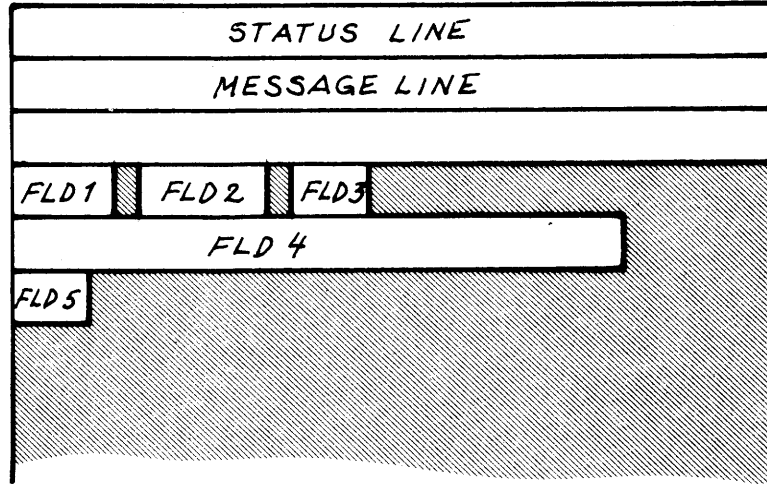
- If there are no more lines available, the next field is keyed from the left most position on the first data line of the screen, which then makes this field the first field on the subformat's next page. Previous to keying this field the screen's input section is blanked.

Example 6.1.1

A subformat starts with the following field descriptions:

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	RTL	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRA
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FLD1				10	..	..	..	..	..	..	..	..	..	..
FLD2				15	..	..	..	..	..	..	..	..	..	..
FLD3				4	..	..	..	..	..	..	..	..	..	..
FLD4				75	..	..	..	..	..	..	..	..	..	..
FLD5				5	..	..	..	..	..	..	..	..	..	..

Keying positions on the screen will subsequently appear as follows:



6.1.2 Establishing Keying Positions

6.1.2

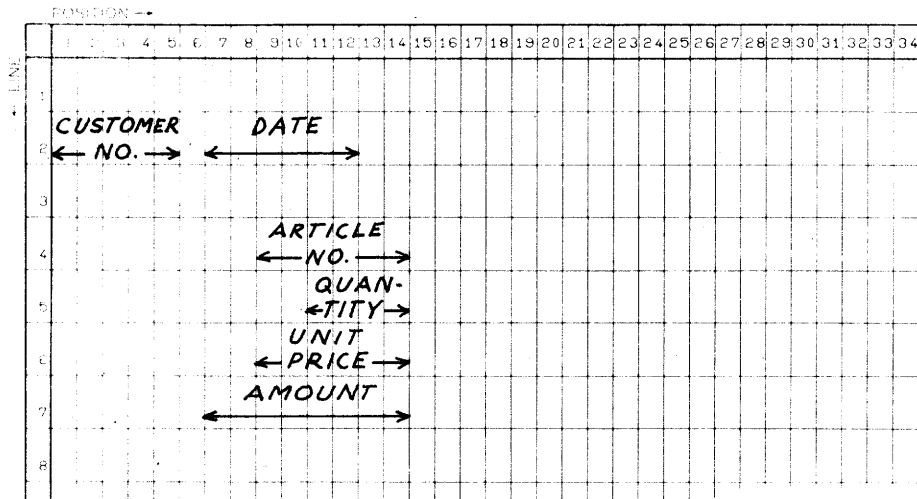
In order to specify where on the screen every single unit of a subformat is to be keyed, the operator fills in: PAGE, LINE, and POSITION in the definition section of the field descriptions. The system will process this information in the following way:

- A subformat's first field will be keyed from the screen position specified under LINE and POSITION in the first field description (PAGE is here, as a rule, specified as 1). This field is the first field on the subformat's first page.
- If PAGE has the same value in the next field description as in the preceding one, the next field is keyed from the screen position specified under LINE and POSITION.
- If PAGE has a higher value in the next field description than in the preceding one, the screen is blanked, and the next field is keyed starting in the screen position specified under LINE and POSITION. This field is then the first field of the subformat's next page.

When planning the keying positions the programmer uses a screen layout form on which he marks out the keying positions and from which the LINE and POSITION values can be read and used when completing the definition sections of the field descriptions.

Example 6.1.2a

Screen layout:



Field descriptions:

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	RL	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STAT
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CUSTM1	2	1	5	..	..	..	..	..	..	..	..	..	..	
DATE	1	2	7	6	..	..	..	..	..	..	..	..	..	
ARTIN1	4	9	6	..	..	..	..	..	..	..	..	..	..	
QTY	1	5	11	4	..	..	..	..	..	..	..	..	..	
UPRIC1	6	9	6	..	..	..	..	..	..	..	..	..	..	
AMOUN1	7	7	8	..	..	..	..	..	..	..	..	..	..	

Within the same subformat it is possible to combine field descriptions with PAGE, LINE, and POSITION filled out, with not completed field descriptions.

Thus, if the system encounters a field description where no information is given, the keying positions will be calculated according to the formula presented in Section 6.1.1. Such field descriptions must therefore be adapted to programmer-selected keying positions, bearing in mind that these might cause the system to shift page. The system does not permit overlapping.

Example 6.1.2b

The field descriptions below will give the same result as the field descriptions presented in Example 6.2.2a (now, PAGE, LINE, and POSITION are not filled out in the DATE column).

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN. LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATI
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CUSTM1	2	1	5	..	..	..	..	..	..	..	..	..	..	
DATE				6	..	..	..	..	..	..	..	..	..	
ARTIM1	4	9	6	..	..	..	..	..	..	..	..	..	..	
QTY	1	5	11	4	..	..	..	..	..	..	..	..	..	
UPRIC1	6	9	6	..	..	..	..	..	..	..	..	..	..	
AMOUN1	7	7	8	..	..	..	..	..	..	..	..	..	..	

6.1.3 Defining Tags

6.1.3

Tags are normally specified only in connection with those formats where the keying positions are defined by the programmer himself (see above). Therefore, tags must be coordinated with keying positions, not only as far as the screen positions are concerned, but also with a view to possible page-shifts in the subformat. Every individual tag within a subformat will then be characterized by PAGE, LINE, and POSITION (see IMAGE Coding Sheet, Section 2.1.2).

The system utilizes the tag information in the following way:

- On selection of a new subformat or a new page within a subformat the screen is blanked and all tags belonging to the new page in the subformat are laid out in their specified screen positions before keying is started to the first field of the page.
- When the first field of a subformat is going to be keyed, the keying positions but not the tags will be blanked if last used tags are the same as those belonging to the first page of the subformat. This is, for instance, the case with subformats which contain only one page and are repeatedly executed.

When devising tags and establishing their position in relation to the keying positions, the programmer uses the screen layout form.

Example 6.1.3a

Extending Example 6.1.2a to include tags gives the following screen layout:

		POSITION →																																	
LINE		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	
1	NUMBER DATE																																		
2	CUSTOMER DATE																																		
	← NO. → ← DATE →																																		
3																																			
4	ARTICLE ← ARTICLE NO. →																																		
5	QUANTITY ← QUANTITY →																																		
6	PRICE ← UNIT PRICE →																																		
7	AMOUNT ← AMOUNT →																																		

Tag specifications:

	PAGE	LINE	POSITION	TEXT
1	2	3	4	
1	1	1		NUMBER DATE
1	4	1		ARTICLE
1	5	1		QUANTITY
1	6	1		PRICE
1	7	1		AMOUNT



Example 6.1.3b

This example shows how a format, at the start of registration, can display a screen image which serves as a keying instruction to the operator.

The format's first subformat might look as follows:

FORMAT NAME		S	P	COMMENT										
1		2	3	4										
<i>EXAMP1N THE SUBFORMAT IS USED AS A GUIDE</i>														
FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	1	6	1	1	O	A	N	O						
														<i>SELECT SUBFORMAT 2, END SUBFORMAT,</i>

Matching tags:

FORMAT NAME		S	COMMENT											
1		2	3											
<i>EXAMP1 GUIDE</i>														
PAGE	LINE	POSITION	TEXT											
1	2	3	4											
1	1	1	<i>FORMAT EXAMP.</i>											
1	3	1	<i>THE FORMAT IS USED FOR KEYING INVOICES</i>											
1	5	1	<i>SUBFORMAT 1: SHOWS THIS GUIDE</i>											
1	6	1	<i>(MAY BE ACTIVATED DURING KEYING, TOO).</i>											
1	7	1	<i>SUBFORMAT 2: HEAD OF INVOICE, SELECTED BY THE OPERATOR</i>											
1	8	1	<i>AT THE START OF A NEW INVOICE.</i>											
1	9	1	<i>SUBFORMAT 3: LINES OF INVOICE, AUTOMATICALLY SELECTED</i>											
1	10	1	<i>AFTER SUBFORMAT 2.</i>											
1	11	1	<i>SUBFORMAT 4: END OF BATCH, SELECTED BY THE OPERATOR</i>											
1	12	1	<i>AFTER LAST INVOICE.</i>											
1	14	1	<i>PRESS THE ENTER KEY WHEN YOU WANT TO START KEYING.</i>											
1	15	1	<i>(THE KEYING WILL START IN SUBFORMAT 2, AUTOMATICALLY).</i>											

The record that the operator creates when pressing the ENTER key contains only one field, a so-called no-transfer field (OUTPUT POSITION = 0). Such a field will be skipped when the batch is transferred to a main computer.

The following final notes on the subject of tags should be added:

1. No tags are attached to a format until the format is entered into the system by activating the supervisor program TRANSLATE (see Chapter 5). When using this format later on, the operator may, however, at the start of registration, command the registration to be executed without tags. (See Users Guide on Control Commands!)
2. When dealing with subformats that entirely rely on system-established keying positions, the available space for tags is limited to what might be left of the lower section of the screen. With such subformats, one may use, for example, DISPLAY statements as a primitive form of tags.

## 6.2 Reformatting

6.2

Reformatting means storing the units of a document in a different order than the keying order. Reformatting is done by specifying suitable values for 'OUTPUT POSITION' in the definition section of the field descriptions.

### Example 6.2

A document contains, among others, the following information:

date: <input type="text"/>	customer no.: <input type="text"/>	article no.: <input type="text"/>	quantity: <input type="text"/>
----------------------------	------------------------------------	-----------------------------------	--------------------------------

The information is typed from left to right, but the preferred storing sequence on the record might be the following:

Field 1	Field 2	Field 3	Field 4
article no.	customer no.	date	quantity

The field descriptions are therefore filled out as follows:

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MAX LENGTH	TYPE	OUTPUT POSITION	RL	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DATE	.	.	.	.	.	.	3	.	.	.	.	.	.	
CUSTN	.	.	.	.	.	.	2	.	.	.	.	.	.	
ARTIN	.	.	.	.	.	.	1	.	.	.	.	.	.	
QTY	.	.	.	.	.	.	4	.	.	.	.	.	.	

The programmer should be careful when specifying output position in order to avoid "gaps" in the subformat (leaving, for example, output position 2 empty, while defining output positions 1 and 3).

**6.3 Automatic Insertion 6.3**

**6.3.1 Not Keyed Fields 6.3.1**

A not keyed field does not require keying, but is assigned its value solely by the program part of the field description.

Such fields can be used to insert, for example, constant values, or results of calculations, or the contents of a register, as values in the record.

Example 6.3.1

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	REKEY DISPLAY	ALIND	REGISTER	PROGRAM STATEMENTS	
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DTYPE				2	2	A	1					N		MOVE 'AA' TO DTYPE, NOTE TYPE OF DOCUMENT,
...														
TOTAL				10	10	N	12	0				N		COMPUTE TOTAL = TOTAL + AMOUNT,
...														
DATE				6	6	N	17					N		COMPUTE DATE = X05,
...														

Note especially the field description "DATE". The register X05 may have been assigned a value at the outset of registration of a specific sequence (typically a subformat that is only used once in a sequence). The date is thereby made available to all subformats during registration. One should take care to control that the register is not used for other programs, for instance, appearing in 'REGISTER' in the definition section of a field description.

6.3.2 Constant Fields

6.3.2

A constant field is assigned the value of a register when the operator activates the DUPLICATE key. The contents of the register are initialized by the program part of a field description.

When a constant field is executed the first time, it is essential that the register specified in the definition section of the field description be assigned a value prior to execution, or it will not be possible for the operator to press the DUP key. This is frequently done at the start of the registration of a specific sequence in subformats that are executed only once within the sequence. Where the contents of the register are preserved throughout the registration of a sequence, they may not be used for other purposes.

If the operator does not use the DUP key for a constant field, the keying takes place as normal. Such keying does not change the contents of the register.

### Example 6.3.2

Subformat 1:

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN. LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DATE				6	6	N	1							COMPUTE X02 = DATE, MOVE ' ' TO X03,

Subformat 2:

DATE				6	6	N	12							C02
CODE				4	1	AN	17							C03

The value of the DATE field in Subformat 2 will be the value registered in Subformat 1, and the CODE field is filled with spaces when pressing the DUP key at these two fields.

## 6.4 Automatic Duplication

6.4

Duplication means assigning a certain field the same value as for the corresponding field in the preceding record.

Duplication is accomplished by utilizing duplication fields.

A duplication field is one that is assigned the value of a register when the operator presses the DUP key. If, by contrast, the operator uses normal keying routines, the keyed data will be input to both field and register.

The DUP key cannot be used with duplication fields, if the specified register has not yet been assigned any value; this is the normal case when first exe-

cuting a duplication field, and the operator must therefore key the field input, which by this action also will be stored in the register.

The use of a register should be limited to one duplication field in a subformat; the register should not be changed by program statements.

Example 6.4

A document is filled out as follows:

CUSTOMER NO	ARTICLE NO	QUANTITY
5002	30	12
5002	992	1
3111	992	97
3111	992	33
4001	30	14

The matching subformat:

FORMAT NAME	S	P	COMMENT
EXAMP2			4

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	RL	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CUSTN				4	4	N	1					DO.1		
ARTIN				3	2	N	2					DO.2		
AMOUN				5	1	N	3	0						END SUBFORMAT,

The effect of DUP on the fourth line, first field, will be the input of 3111 as customer number in the record, if this number is keyed in the corresponding position in the third record.

## 6.5 Automatic Incrementation

6.5

Automatic incrementation means that a field is assigned the value of some previously keyed value increased by 1.

Automatic incrementation uses increment fields. An increment field has the same function as a duplication field (see Section 6.4), except that the corresponding register is increased by 1 by activating the DUP key. The keying of an increment field causes also the corresponding register to be changed.

### Example 6.5

A document is filled out as follows:

TYPE	INFORMATION 1	INFORMATION 2
1	1250	20
2	900	1992
3		
4	11	77

The matching subformat:

FORMAT NAME	S	P	COMMENT
1	2	3	4

*EXAMP2*

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
<i>TYPE</i>				<i>1</i>	<i>1</i>	<i>N</i>	<i>1</i>							
<i>INF1</i>				<i>4</i>	<i>1</i>	<i>N</i>	<i>2</i>							
<i>INF2</i>				<i>4</i>	<i>1</i>	<i>N</i>	<i>3</i>							<i>END SUBFORMAT,</i>

On the first line of the field TYPE '1' is keyed, on the next DUP, the third line is left empty, and on the fourth '4' is keyed.

Tables are used in programs during calculations when an operand may contain one out of so many values that a comparison of the current value with every single possible value, by program statements, would be an insurmountable task (as, for instance, using IF and ALLOW/DISALLOW with constants representing allowed/disallowed values).

Therefore, the values are gathered together in a table and can thus be referenced collectively by a single program statement.

The use of tables ensures another advantage in that they can be utilized by different programs.

There are two types of tables:

1. If one only wants to find out if a certain value exists, a single-entried table is used, listing all possible values.
2. If the operation not only involves establishing the existence of a certain value (argument), but also the access to an associated value (function), a multiple-entried table is used, which contains all possible arguments together with their assigned values.

When used for registration, a format is taken from the disc and placed in the internal core together with the referenced tables. In the case of very long tables, it is therefore recommended to create so-called DISC tables, in order to save space in the computer. A DISC table will always be stored on the disc only. Please also notice that core tables cannot contain more than two columns (single-entried or double-entried) where as DISC tables can contain up to seven columns (six functions per argument). See Chapter 5 for further details!

#### Example 6.6

A document contains, among other things, the following units:

MONTH:	<input type="text"/>	DAY:	<input type="text"/>	DATE:	<input type="text"/>
--------	----------------------	------	----------------------	-------	----------------------



Corresponding field descriptions:

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	PK	FILL	REPLY	DISPLAY	MODE	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MONTH				9	1	AN	10L							IF NOT (MONTH IN MTABL) THEN ALARM 'MONTH IS WRONG',
DAY				9	1	AN	11L							IF NOT (DAY IN DTABL) THEN ALARM 'DAY IS WRONG',
DATE				2	1	N	12							IF DATE < 1 THEN ALARM 'DATE IS WRONG', IF MONTH VALID THEN SEARCH MONTH IN MTABL GIVING X01 AT END ALARM 'SYSTEM ERROR', ELSE COMPUTE X01 = 31, IF DATE > X01 THEN ALARM 'DATE IS WRONG',

Two tables are referenced:

one single-entried and one double-entried

TABLE NAME	T	A TYPE	A LGTH	F TYPE	F LGTH
1	2	3	4	5	6
DTABL	A	N	9		

1 ARGUMENTS	
1	MONDAY
2	TUESDAY
1	WEDNESDAY
2	THURSDAY
1	FRIDAY
2	SATURDAY
1	SUNDAY

TABLE NAME	T	A TYPE	A LGTH	F TYPE	F LGTH
1	2	3	4	5	6
MTABL	D	A	N	9	2

1 & 2 ARGUMENTS & FUNCTIONS	
1	JANUARY
2	31
1	FEBRUARY
2	29
1	MARCH
2	31
...	
1	NOVEMBER
2	30
1	DECEMBER
2	31

Note that MTABLE in 'MONTH' actually serves as a single-entried table.

If MTABL were a disctable (though inconvenient with respect to its size) instead of a core table then the following changes had to be made in the program statements in above field descriptions:

1. The statement in MONTH must be replaced by the following (disctable not allowed in table condition):

SEARCH MONTH IN DISC MTABL GIVING \*  
AT END ALARM 'MONTH IS WRONG',

2. The fourth line of DATE must be replaced by:

SEARCH MONTH IN DISC MTABL

The table would be coded like this:

TABLE NAME	T	A	F	S	NO. OF F'S	A- TYPE	F1- TYPE	F2- TYPE	F3- TYPE	F4- TYPE	F5- TYPE	F6- TYPE
1	2				7	8	9	10	11	12	13	14
M T A B L	M				1	A	N	N				

OP	ONLY TO BE KEYED IF FILLED IN. OP = I, U, D, E.
1	ARGUMENTS & FUNCTIONS
2	
i	J A N U A R Y
	3 1
i	F E B R U A R Y
	2 9
i	M A R C H
	3 1
	...
i	N O V E M B E R
	3 0
i	D E C E M B E R
	3 1
E	

## 6.7 Partial Rekeying

6.7

In order to save time when rekeying it might often be appropriate to rekey only certain units of certain documents.

This procedure is controlled by the column 'REKEY' in the definition sections of the field descriptions. If the REKEY column is keyed N (for No), the corresponding field is skipped when rekeying.

As a rule, fields containing important information will always be rekeyed, if the information can not be thoroughly checked at the initial keying stage by the use of tables, LIMIT statements, or the like. Take, for example, amounts included in a sum total: if the total turns out to be false, one does not know if the keying error occurred when the total was keyed or is hidden among the amounts added up.

## 6.8 The Use of Pseudo-Registers

The pseudo-registers described in section 3.3.2 enable a format to read out information about the environment as it looks when the format is used for registration.

The information may for example be stored in the batch for trace purposes. Consider a format which, at the start of a registration, outputs name of job and name of batch together with the time of registration start-up. Once stored the information will also be available to the computer to which the batch is transferred for further processing.



# Appendix I

## Required Space In Core For Formats, Subprograms and Tables

The following sizes are all defined in bytes. In some cases length is defined as an interval (lower limit - upper limit); a definition section may, for instance, at best fill 15 bytes and at worst 16 bytes.

format head	16 + 4 * number of subformats
subformat head	8
definition section	15-16
image head	4 + 4 * number of subformats
image subformat head	4 + 2 * number of pages
image page	1 + 5 * number of texts + length of texts
<u>Program part</u>	
field reference	3
register reference	2
constant	2 + constant length
subscript	1
table (not DISC table) (1st reference of table)	1-2 + table length
table (not DISC table) (subsequent references)	3
DISC table	34-35
label reference	3-4
subformat reference	2
translated subprogram	as for normal program part
translated table (not DISC table)	6 + number of arguments * (length of one argument + length of one function)

+, -, *, /, MOD, <>, =, <, >, <=, >=	1
AND, OR, NOT, VALID, INVALID, IN	1
COMPUTE - =	1
MOVE - TO	1
CONNECT - TO - GIVING	2
SEARCH - GIVING - AT END	9-11
LIMIT	1
ALLOW	1-3
DISALLOW	1-3
GOTO	1
SELECT SUBFORMAT	1
DISPLAY	1
ALARM	1
DEFINE - length	3
END	1
END SUBFORMAT	0
NOTE	0
PERFORM (1st reference of subprogram)	9-12 + subprogram length
PERFORM (subsequent references)	4-5
DUP-FIELDS	3
SKIP - FIELDS	3
IF - THEN	4-5
IF - THEN - ELSE	8-10
SET	1

Study the example on the opposite page!

FORMAT NAME	3	4	COMMENT
EXPL 1	1		REQUIRED SPACE - 1

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	DEFINITION POSITION	FL	FN	BEHAV	DISPLAY	END	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
F1				1	0	N	1							DEFINE X01 1, COMPUTE X01=F1, PERFORM CHECK,
				0										SELECT SUBFORMAT 2, END SUBFORMAT,

FORMAT NAME	3	4	COMMENT
EXPL 2	2		REQUIRED SPACE - 2

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	DEFINITION POSITION	FL	FN	BEHAV	DISPLAY	END	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				0										DISPLAY X01,
F1				1	0	N	0							COMPUTE X01=F1, PERFORM CHECK,
				0										END,

Translated, this format requires the following number of bytes:

format head (16 + 4 \* 2)

24

subformat 1:

subformat head

8

1st field description:

definition section

15-16

DEFINE - 1

3

X01 (register reference)

2

COMPUTE - =

1

X01

2

F1 (field reference)

3

PERFORM CHECK\*) (9 → 12 + 50) 59-62 85-89

\*) The subprogram CHECK is assumed to require 50 bytes.

2nd field description:			
definition section	15-16		
SELECT SUBFORMAT	1		
2 (subformat reference)	2		
END SUBFORMAT	<u>0</u>	<u>18-19</u>	111-116
subformat 2:			
subformat head		8	
1st field description:			
definition section	15-16		
DISPLAY	1		
X01	<u>2</u>	18-19	
2nd field description:			
definition section	15-16		
COMPUTE - =	1		
X01	2		
F1	3		
PERFORM CHECK	<u>4-5</u>	25-27	
3rd field description:			
definition section	15-16		
END	<u>1</u>	<u>16-17</u>	<u>67-71</u>
Total number of bytes:			<u>202-211</u>

In other words, the translated format will require something between 202 and 211 bytes.



# Appendix II

## Required Space On Disc For Batches

The control command SET (see Users Guide for further information) reserves a number of disc segments for a batch. Each segment holds 512 bytes.

During the keying operation three kinds of data are stored in the batch:

1. Batch description,
2. Data records,
3. Register records.

The following describes how to calculate required space for records that are entered with a given format. (All sizes in bytes.)

### 1. Batch Description

Fixed length of 1 segment = 512 bytes.

### 2. Data Records

Gross record length calculated as:

the sum of defined field lengths  
+ 2 \* number of fields in the subformat (including not keyed and  
0-length fields)  
+ 13

### 3. Register Records

In order to facilitate replay a picture of the registers - a register record - is frequently kept in the batch.

The length of the register record is calculated as follows:

the sum of defined register lengths  
+ 4 \* largest defined register index  
+ 17

One register record is stored for each 10 data records.

Example

Consider a format with only one subformat which contains 10 fields and has a net field length of 60 characters.

Gross record length =  $60 + 2 * 10 + 13 = 93$  characters.

If the format uses X01-X03 with a total length of 20 characters, then the register record will hold

$$20 + 4 * 3 + 17 = 49 \text{ characters}$$

If 500 documents of this type are to be registered, the batch will require the following disc space:

$$512 + 500 * 93 + (500/10) * 49 = 49462 \text{ characters} = 97 \text{ segments.}$$

# Appendix III

## Examples

### A Format Example

As an example of a format with image, see the invoice from a greengrocer on page A III - 2.

The invoice consists of three parts:

- 1) The head of the invoice with fields concerning the customers.
- 2) The body with one line for each sort of vegetables the customer has bought.
- 3) The end of the invoice with the total price.

The format consists of four subformats:

H - defines the registers used.

- 1 - controls input of the head of the invoice.
- 2 - controls input of one line in the body of the invoice.
- 3 - controls input of the end of the invoice.

170950-0712	Discount percentage 50	Invoice number 45711
1. Customer number		
2. Customer name L. Brown		
3. Customer address 39 Main Street		
3000	5. City Appleville	Date 76.09.03
4. Postal code		

1. Article no.	2. Article name	3. Quantity	4. Unit price	Discount	5. Final price
1010101010	Apple	3	5.000	7.500	7.500
1010101060	Pear	3	4.000	6.000	6.000

13.500
1. Total price



# SCREEN LAYOUT

NOTES: HEAD OF INVOICE

PAGE 1 OF 3

INITIALS: AA

DATE: 76.09.03

FORMAT: /INVOI

SUBFORMAT: 1

PAGE 1

POSITION: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

CUSTOMER NUMBER  
XXXXXXXXXXXX

CUSTOMER NAME  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

CUSTOMER ADDRESS  
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

POSTAL CODE  
XXXX

CITY  
XXXXXXXXXXXXXXXXXXXX



# SCREEN LAYOUT

NOTES ONE LINE IN BODY OF INVOICE PAGE 2 OF 3

INITIALS AA

DATE 76.09.03

FORMAT: INV01

SUBFORMAT: 2

PAGE 1

POSITION	ART NUMBER	ARTICLE NAME	QUANTITY	UNIT PRICE	DISCOUNT	FINAL PRICE
1	XXXXXXXXXX	XXXXXXXXXXXXXXXXXXXX	XXXXXX	XXXXXXXXXX	XXXXXXXXXX	XXXXXXXXXX
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						
21	SELECT SUBFORMAT 3 WHEN THE BODY OF INVOICE IS FINISHED					



# FORMAT CODING SHEET

NOTES

INITIALS: AA

DATE: 76.09.03

FORMAT: /INVOI

FORMAT NAME: 1

LINE: 1

POSITION: 1

LENGTH: 1

MM LENGTH: 1

TYPE: 1

OUTPUT POSITION: 1

R/L: 1

FILL: 1

KEY: 1

DISPLAY: 1

KIND: 1

REGISTER: 1

PROGRAM STATEMENTS: 1

## INVOIHSUBFORMAT H - DEFINITION OF REGISTERS

LINE	POSITION	LENGTH	MM LENGTH	TYPE	OUTPUT POSITION	R/L	FILL	KEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	1	0										DEFINE X01 3, NOTE DISCOUNT PERCENTAGE, DEFINE X02 10, NOTE ARTICLE NUMBER, DEFINE X03 10, NOTE FINAL PRICE, DEFINE X04 15, NOTE TOTAL PRICE, DEFINE X05 10, NOTE REGISTER FOR SEARCHING ARTICLE, SELECT SUBFORMAT 1, END SUBFORMAT,





# FORMAT CODING SHEET

FORMAT INVOI

DATE: 76.09.03

INITIALS: AA

NOTES:

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	R/L	KEY					REGISTER	PROGRAM STATEMENTS
									11	12	13	14	15		
INVOI1	1	1	1	12	8	AN	1L								NOTE READ CUSTOMER NUMBER - SEARCH IN CUSTOMER TABLE FOR DISCOUNT PERCENTAGE AND DUMP IT, SEARCH CUSNO IN CTABLE GIVING XO1 AT END ALARM 'CUSTOMER NUMBER NOT KNOWN',
DISCT				3	ON		2								NOTE INSERT DISCOUNT PERCENTAGE IN RECORD, IF CUSNO VALID THEN COMPUTE DISCT= XO1 ELSE COMPUTE XO1= 0,
NAME	1	2	20	50	1	A	3L								NOTE READ NAME,
ADDR	1	5	150		1	AN	4L								NOTE READ ADDRESS,
PCODE	1	8	1	4	4	N	5								NOTE READ POSTAL CODE, LIMIT 1000 9000, NOTE LIMITS 1000 <= PCODE <= 9000,
CITY	1	8	20	18	1	A	6L								NOTE READ CITY,

	FORMAT CODING SHEET	NOTES:	PAGE 3 OF 6
INITIALS <b>AA</b>	DATE <b>76.09.03</b>	FORMAT <b>/NVOI</b>	

FORMAT NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	RL	FILL	REKEY	DISPLAY	REGISTER KIND	REGISTER	PROGRAM STATEMENTS	
														1	2
1														15	NOTE DUMMY FIELD FOR RESETTNG REGISTER, COMPUTE X04= 0, SELECT SUBFORMAT 2, END SUBFORMAT,



# FORMAT CODING SHEET

PAGE 4 OF 6

FORMAT /NVOI

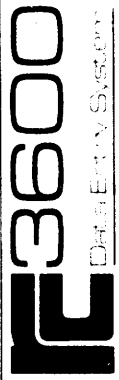
DATE 76.09.03

INITIALS AA

NOTES

INVOI 2 SUBFORMAT 2 - ONE LINE IN BODY OF INVOICE

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN LENGTH	TYPE	OUTPUT POSITION	RL	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ARTNO1	2	1	10	10			1							NOTE READ ARTICLE NUMBER AND PERFORM CHECK DIGIT VERIFICATION (MODULUS 10), COMPUTE X02= ARTNO, PERFORM CHE10, IF ARTNO = 0 THEN SKIP 1 FIELD,
NAME	1	2	13	20	1	A	2	L						NOTE READ ARTICLE NAME - SEARCH IN ARTICLE TABLE FOR CONNECTION BETWEEN NUMBER AND NAME, SEARCH NAME IN ATABL GIVING X05 AT END ALARM 'ARTICLE NAME NOT KNOWN', IF X05 <> ARTNO THEN ALARM 'ARTICLE NUMBER AND NAME DO NOT MATCH',
QTY	1	2	3	5	6	1	N	3						NOTE READ QUANTITY,
UNIT	1	2	4	5	10	1	N	4						NOTE READ UNIT PRICE,



# FORMAT CODING SHEET

PAGE 5 OF 6

INITIALS AA

DATE 76.09.03

FORMAT: /NVOI

NOTES:

LINE	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
FILE NAME															
LINE															
TYPE															
MIN. LENGTH															
LENGTH															
POSITION															
TYPE															
OUTPUT POSITION															
FILE															
KEY															
DISPLAY															
KIND															
REGISTER															
PROGRAM STATEMENTS															

1	DISCT1	25710	ON	5	Y	N														
<p>NOTE COMPUTE PRICE AND DUMP IT -                  COMPUTE DISCOUNT AND INSERT IN RECORD,                  COMPUTE X03= QTY * UNITP, COMPUTE DISCT= X01 * X03 / 100,                  NOTE READ FINAL PRICE AND CHECK AGAINST COMPUTED PRICE,                  IF FINAL &lt;&gt; X03 - DISCT THEN                  ALARM 'FINAL PRICE NOT OK',                  NOTE UPDATE TOTAL PRICE,                  COMPUTE X04= X04 + FINAL,                  END SUBFORMAT,</p>																				
2																				
3																				
4																				
5																				
6																				
7																				
8																				
9																				
10																				
11																				
12																				
13																				
14																				
15																				



# FORMAT CODING SHEET

CHEMICAL SYSTEMS

INITIALS: AA

DATE: 76.09.03

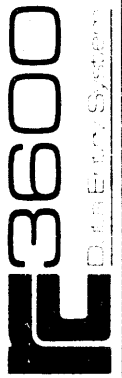
NOTES:

PAGE 6 OF 6

FORMAT: /NVO/

INVO/3 SUBFORMAT 3 - END OF INVOICE

LINE NO.	NO.	TIME	POSITION	LENGTH	IN. LENGTH	TYPE	OUTPUT POSITION	RT	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STATEMENTS
TOTAL	1	264	15	1	N		1							NOTE READ TOTAL PRICE AND CHECK IT AGAINST TOTALLED UP PRICE, IF TOTAL <> X04 THEN ALARM 'TOTAL PRICE NOT OK',
MORE	121	64	3	2	A	OL								NOTE READ CONTINUATION - YES : MORE TO BE KEYED - NO : END FORMAT, ALLOW 'YES', 'NO', IF MORE = 'YES', THEN SELECT SUBFORMAT 1; NOTE SELECT HEAD OF INVOICE, END;



# IMAGE CODING SHEET

INITIALS

AA

DATE

76.09.03

NOTES

IMAGE TO HEAD OF INVOICE

PAGE 1 OF 3

FORMAT:

INVOI

LINE	QTY	POSITION	TEXT	COMMENT
1	1		1	1
1	1		1	1
1	4		1	4
1	7		1	7
1	7		7	7

INVOI 1 IMAGE TO HEAD OF INVOICE

1 CUSTOMER NUMBER  
 1 20 CUSTOMER NAME  
 1 4 1 CUSTOMER ADDRESS  
 1 7 1 POSTAL CODE  
 1 7 20 CITY



# IMAGE CODING SHEET

INITIALS

AA

DATE:

76.09.03

NOTES:

IMAGE TO ONE LINE IN BODY

PAGE 2 OF 3


FORMAT

2

LINE	POS.	TEXT
1	1	ART NUMBER
1	2	ARTICLE NAME
1	3	QUANTITY
1	4	UNIT PRICE
1	5	DISCOUNT
1	6	FINAL PRICE

INVO/2/IMAGE TO ONE LINE IN BODY OF INVOICE

1 21 1SELECT SUBFORMAT 3 WHEN THE BODY OF INVOICE IS FINISHED

		<b>IMAGE CODING SHEET</b>	NOTES:	PAGE 3 OF 3
INITIALS <b>AA</b>	DATE <b>76.09.03</b>	IMAGE TO END OF INVOICE		
FORMAT NAME 1	COMMENT 2	FORMAT <b>/NVOI</b>		

INVOI3 / IMAGE TO END OF INVOICE		
FORMAT NAME 1 2 3	COMMENT 2 3	INITIALS <b>AA</b>
DATE <b>76.09.03</b>	NOTES:	

LINE	POSITION	TEXT
1	1	64 TOTAL PRICE
1	20	1 KEY YES IF YOU WANT TO START ON A NEW INVOICE
1	21	1 KEY NO IF YOU WANT TO FINISH KEYING





SUBPROGRAM CODING SHEET

INITIALS: AA

DATE: 76.09.03

NOTES:

SUBPROGRAM USED IN INVOICE

PAGE 1 OF 1

SUBPROGRAM: CHE10

SUB-PROG NAME

1

COMMENT:

2

CHE10SUBPROGRAM - CHECK DIGIT VERIFICATION (MODULUS 10) - X02 = ARTICLE NUMBER

PROGRAM STATEMENTS

1

COMPUTE X02 = (X02(1) \* 2

+ X02(2)

+ X02(3) \* 2

+ X02(4)

+ X02(5) \* 2

+ X02(6)

+ X02(7) \* 2

+ X02(8)

+ X02(9) \* 2

+ X02(10) MOD 10,

IF X02 <> 0 THEN ALARM 'ERROR IN ARTICLE NUMBER',

END,



TABLE CODING SHEET (DOUBLE)

INITIALS: AA

DATE: 76.09.03

NOTES:

ARTICLE TABLE USED IN INVOICE

PAGE 1 OF 1

TABLE: ATABL

TABLE NAME	T	A TYPE	A LGTH	F TYPE	F LGTH
1	2	3	4	5	6
ATABL	D	AN	20	N	10

1 & 2 ARGUMENTS & FUNCTIONS

1 POTATO

2 0

1 APPLE

2 1010101010

1 PEAR

2 1010101060

1 ONION

2 1234567890

1 HORSE-RADISH

2 2345678906

1

2

1

2

1

2

1

2

1

2



**TABLE CODING SHEET (DOUBLE)**

INITIALS: AA

DATE: 76.09.0

PAGE 1 OF 1

TABLE CTABL

NOTES: CUSTOMER TABLE USED IN INVOICE

TABLE NAME	T	A	A	F	F
	1	2	3	4	5
	TYPE	LGTH	TYPE	LGTH	
CTABL	D	AM	12M		3

1 & 2 ARGUMENTS & FUNCTIONS

1 2841-201

2 0

1 280149-2323

2 10

1 170950-0712

2 50

1 5482-170

2 25

1 3567-8111

2 33

1 00000000

2 0

1 147680-9221

2 3

1 RC

2 100

1

2

1

2

LIST: FORMAT=SOURCE TEXT (EXBU1):

REC FNAME S P COMMENT  
0001 INVOI H Y SUBFORMAT H - DEFINITION OF REGISTERS

```

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
H 001 001 0002          U          DEFINE X01 3, NOTE DISCOUNT PERCENTAGE,
H 001 002 0003          U          DEFINE X02 10, NOTE ARTICLE NUMBER,
H 001 003 0004          U          DEFINE X03 10, NOTE FINAL PRICE,
H 001 004 0005          U          DEFINE X04 15, NOTE TOTAL PRICE,
H 001 005 0006          U          DEFINE X05 10, NOTE REGISTER FOR SEARCHING ARTICLE,
H 001 006 0007          U          SELECT SUBFORMAT 1,
H 001 007 0008          U          END SUBFORMAT,

```

REC FNAME S P COMMENT  
0009 INVOI 1 SUBFORMAT 1 - HEAD OF INVOICE

```

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
1 001 001 0010 CUSNO 1 2 1 12 8 AN 1 L NOTE READ CUSTOMER NUMBER - SEARCH IN CUSTOMER TABLE FOR
1 001 002 0011          U          DISCOUNT PERCENTAGE AND DUMP IT,
1 001 003 0012          U          SEARCH CUSNO IN CTABL GIVING X01 AT END,
1 001 004 0013          U          ALARM 'CUSTOMER NUMBER NOT KNOWN',
1 001 005 0014          U          NOTE INSERT DISCOUNT PERCENTAGE IN RECORD,
1 002 001 0015 DISCT          3 0 N 2 N IF CUSNO VALID THEN COMPUTE DISCT= X01 ELSE COMPUTE X01= 0,
1 002 002 0016          U          NOTE READ NAME,
1 002 003 0017          U          NOTE READ ADDRESS,
1 003 001 0018 NAME 1 2 20 5U 1 A 3 L
1 003 002 0019          U          NOTE READ POSTAL CODE,
1 004 001 0020 ADDR 1 5 1 5U 1 AN 4 L
1 004 002 0021          U          LIMIT 1000 9000, NOTE LIMITS 1000 <= PCODE <= 9000,
1 005 001 0022 PCODE 1 8 1 4 4 N 5
1 005 002 0023          U          NOTE READ CITY,
1 005 003 0024          U          NOTE DUMMY FIELD FOR RESETTING REGISTER,
1 006 001 0025 CITY 1 8 20 18 1 A 6 L
1 006 002 0026          U          COMPUTE X04= 0,
1 007 001 0027          U          SELECT SUBFORMAT 2,
1 007 002 0028          U          END SUBFORMAT,
1 007 003 0029          U
1 007 004 0030          U

```

REC FNAME S P COMMENT  
0031 INVOI 2 SUBFORMAT 2 - ONE LINE IN BODY OF INVOICE

```

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
2 001 001 0032 ARTNO 1 2 1 10 10 N 1 0 NOTE READ ARTICLE NUMBER AND PERFORM CHECK DIGIT
2 001 002 0033          U          VERIFICATION (MODULUS 10),
2 001 003 0034          U          COMPUTE X02= ARTNO, PERFORM CHE10,
2 001 004 0035          U          IF ARTNO = 0 THEN SKIP 1 FIELD,
2 001 005 0036          U          NOTE READ ARTICLE NAME - SEARCH IN ARTICLE TABLE FOR
2 002 001 0037 NAME 1 2 13 2U 1 AN 2 L CONNECTION BETWEEN NUMBER AND NAME,
2 002 002 0038          U          SEARCH NAME IN ATABL GIVING X05 AT END,
2 002 003 0039          U          ALARM 'ARTICLE NAME NOT KNOWN',
2 002 004 0040          U          IF X05 <> ARTNO THEN
2 002 005 0041          U          ALARM 'ARTICLE NUMBER AND NAME DO NOT MATCH',
2 002 006 0042          U          NOTE READ QUANTITY,
2 002 007 0043          U          NOTE READ UNIT PRICE,
2 003 001 0044 QTY 1 2 35 6 1 N 3
2 003 002 0045          U          NOTE COMPUTE PRICE AND DUMP IT =
2 004 001 0046 UNITP 1 2 45 10 1 N 4 COMPUTE DISCOUNT AND INSERT IN RECORD,
2 004 002 0047          U          COMPUTE X03= QTY * UNITP, COMPUTE DISCT= X01 * X03 / 100,
2 005 001 0048 DISCT 1 2 57 10 0 N 5 Y N
2 005 002 0049          U          NOTE READ FINAL PRICE AND CHECK AGAINST COMPUTED PRICE,
2 005 003 0050          U          IF FINAL <> X03 = DISCT THEN
2 005 004 0051          U          ALARM 'FINAL PRICE NOT OK',
2 006 001 0052 FINAL 1 2 69 10 1 N 6 NOTE UPDATE TOTAL PRICE,
2 006 002 0053          U          COMPUTE X04= X04 + FINAL,
2 006 003 0054          U          END SUBFORMAT,
2 006 004 0055          U
2 006 005 0056          U
2 006 006 0057          U

```

REC FNAME S P COMMENT  
0058 INVOI 3 SUBFORMAT 3 - END OF INVOICE

```

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
3 001 001 0059 TOTAL 1 2 64 15 1 N 1 NOTE READ TOTAL PRICE AND CHECK IT AGAINST TOTALLED UP PRICE,
3 001 002 0060          U          IF TOTAL <> X04 THEN
3 001 003 0061          U          ALARM 'TOTAL PRICE NOT OK',
3 001 004 0062          U          NOTE READ CONTINUATION - YES : MORE TO BE KEYED - NO : END FORMAT,
3 002 001 0063 MORE 1 21 64 3 2 A 0 L ALLOW 'YES' 'NO',
3 002 002 0064          U          IF MORE = 'YES' THEN
3 002 003 0065          U          SELECT SUBFORMAT 1; NOTE SELECT HEAD OF INVOICE,
3 002 004 0066          U          END,
3 002 005 0067          U

```

END LIST

LIST: SUBPROGRAM=SOURCE TEXT (EXB04):

REC SNAME COMMENT  
0001 CHE10 SUBPROGRAM - CHECK DIGIT VERIFICATION (MODULUS 10) - X02 = ARTICLE NUMBER

LIN REC PROGRAM STATEMENTS

```

001 0002 COMPUTE X02= (X02(1))*2
002 0003          +X02(2)
003 0004          +X02(3)*2
004 0005          +X02(4)
005 0006          +X02(5)*2
006 0007          +X02(6)
007 0008          +X02(7)*2
008 0009          +X02(8)
009 0010          +X02(9)*2
010 0011          +X02(10) MOD 10,
011 0012 IF X02 <> 0 THEN ALARM 'ERROR IN ARTICLE NUMBER',
012 0013 END,

```

END LIST

LIST: TABLE-SOURCE TEXT (EXB02):

```
REC TNAME T AT AL FT FL
0001 CTABL D AN 12 N 03

ENTRY REC ARGUMENTS AND FUNCTIONS
00001 0002 2841-201
      UUU
00002 0003 280149-2323
      U1U
00003 0004 170950-0712
      U5U
00004 0005 5482-170
      J25
00005 0006 3567-8111
      U35
00006 0007 00000000
      UUU
00007 0008 147680-9221
      U05
00008 0009 RC
      100
```

END LIST

LIST: TABLE-SOURCE TEXT (EXB03):

```
REC TNAME T AT AL FT FL
0001 ATABL D AN 20 N 10

ENTRY REC ARGUMENTS AND FUNCTIONS
00001 0002 POTATO
      UUUUUUUUUUU
00002 0003 APPLE
      1010101010
00003 0004 PEAR
      1010101060
00004 0005 ONION
      1234567890
00005 0006 HORSE-RADISH
      2345678906
```

END LIST

LIST: IMAGE-SOURCE TEXT (EXB05):

```
REC FNAME S COMMENT
0001 INVOI 1 IMAGE TO HEAD OF INVOICE

S REC P LN PS TEXT
1 0002 1 1 1 CUSTOMER NUMBER
1 0003 1 1 20 CUSTOMER NAME
1 0004 1 4 1 CUSTOMER ADDRESS
1 0005 1 7 1 POSTAL CODE
1 0006 1 7 20 CITY

REC FNAME S COMMENT
0007 INVOI 2 IMAGE TO ONE LINE IN BODY OF INVOICE

S REC P LN PS TEXT
2 0008 1 1 1 ART NUMBER ARTICLE NAME QUANTITY UNIT PRICE DISCOUNT FINAL PRICE
2 0009 1 21 1 SELECT SUBFORMAT 3 WHEN THE BODY OF INVOICE IS FINISHED

REC FNAME S COMMENT
0010 INVOI 3 IMAGE TO END OF INVOICE

S REC P LN PS TEXT
3 0011 1 1 64 TOTAL PRICE
3 0012 1 20 1 KEY YES IF YOU WANT TO START ON A NEW INVOICE
3 0013 1 21 1 KEY NO IF YOU WANT TO FINISH KEYING
```

END LIST

TRANS TABLE EXB02

SIZE OF CTABL : 00126 BYTES

TRANS TABLE EXB03

SIZE OF ATABL : 00156 BYTES

TRANS SUBPROGRAM EXB04

SIZE OF CHE10 : 00104 BYTES

TRANS FORMAT WITH IMAGE EXB01 EXB05

SIZE OF IMAGE : 00422 BYTES

SIZE OF INVOI : 01076 BYTES



# Appendix IV

## Standard Formats FORM, IMAGE, SUBPR, and TABLE

LIST: FORMAT-SOURCE TEXT (STD01):

```

REC  FNAME S P COMMENT
0001 FORM  H N  FORMAT  FORMAT  - SUBFORMAT HEAD ENTRY

S  FLD  LIN  REC  NAME  P  LN  PS  LG  NL  TY  OUT  J  F  R  D  K  RG  PROGRAM STATEMENTS
H 001 001 0002 NAME  1  1  17  5  1  AN  1  L          NOTE THE FORMAT NAME IS 1 TO 5 CHARACTERS,
H 001 002 0003          IF (NAME(1)<'A') OR (NAME(1)>'Z') THEN
H 001 003 0004          ALARM 'ILLEGAL FORMAT NAME',
H 002 001 0005 SUBFM  1  2  17  1  1  AN  2          NOTE THE SUBFORMAT NAME IS 1 CHARACTER,
H 002 002 0006          IF NOT((SUBFM>'0') AND (SUBFM<'9')) OR
H 002 003 0007          ((SUBFM='A') AND (SUBFM<'Z')) THEN
H 002 004 0008          ALARM 'ILLEGAL SUBFORMAT NAME',
H 003 001 0009 PROCT  1  3  17  1  0  A  3          NOTE SUBFORMAT PROTECTION AGAINST MANUAL SELECTION;
H 003 002 0010          Y = NO MANUAL PROTECTION AGAINST MANUAL SELECTION;
H 003 003 0011          ALLOW 'Y' 'N' ' ',
H 004 001 0012 CORR  1  5  1  74  0  AN  4  L          NOTE SUBFORMAT HEAD IS FINISHED -
H 005 001 0013          CHANGE SUBFORMAT TO FIELD DESCRIPTION,
H 005 002 0014          DEFINE X02 5, MOVE ' ' TO X02, NOTE CONSTANT = SPACES,
H 005 003 0015          DEFINE X03 1, NOTE KEEPS TRACK OF PAGENO = LINENO AND POS,
H 005 004 0016          DEFINE X04 1, COMPUTE X04 = 1, NOTE LAST USED PAGE,
H 005 005 0017          DEFINE X05 3,
H 005 006 0018          NOTE USED WHEN PAGE = MIN.LENGTH AND OUTPUT POS ARE CHECKED,
H 005 007 0019          SELECT SUBFORMAT F,
H 005 008 0020          END SUBFORMAT,
H 005 009 0021

```

```

REC  FNAME S P COMMENT
0022 FORM  F N  FIELD DESCRIPTION ENTRY

S  FLD  LIN  REC  NAME  P  LN  PS  LG  NL  TY  OUT  J  F  R  D  K  RG  PROGRAM STATEMENTS
F 001 001 0023 FLDNM  1  1  15  5  0  AN  1  L          NOTE FIELD NAME IS 0 TO 5 CHARACTERS,
F 001 002 0024          IF ((FLDNM(1)<'A') OR (FLDNM(1)>'Z')) AND (FLDNM<>X02) THEN
F 001 003 0025          ALARM 'ILLEGAL FIELDNAME',
F 002 001 0026 PAGE  1  1  32  1  0  N  2          NOTE PAGENO IN DISPLAY-LAYOUT = FROM 1 TO 8,
F 002 002 0027          IF ALPHANUMERIC PAGE = ' ' THEN SKIP 2 FIELDS,
F 002 003 0028          LIMIT 1 8,
F 002 004 0029          IF PAGE<X04 THEN ALARM 'CURRENT PAGENO LESS THAN PREVIOUS PAGENO',
F 002 005 0030          COMPUTE X04 = PAGE,
F 003 001 0031 LINE  1  1  45  2  0  N  5  R          NOTE LINENO IN DISPLAY-LAYOUT = FROM 1 TO NO OF LINES ON DISPLAY,
F 003 002 0032          LIMIT 1 21,
F 004 001 0033 POS  1  1  61  2  0  N  4  R          NOTE POSITION IN LINENO DISPLAY-LAYOUT =
F 004 002 0034          FROM 1 TO NO OF POSITIONS PER LINE ON DISPLAY,
F 004 003 0035          LIMIT 1 80,
F 005 001 0036 LENGT  1  2  13  2  0  N  5  R          NOTE THE FIELD LENGTH IS FROM 0 TO 80,
F 005 002 0037          IF LENGTH = 0 THEN SKIP 9 FIELDS,
F 005 003 0038          LIMIT 1 80,
F 006 001 0039 MINLE  1  2  32  2  0  N  6  R          NOTE THE MINIMUM LENGTH OF THE KEYED DATA,
F 006 002 0040          IF MINLE>LENGTH THEN
F 006 003 0041          ALARM 'MIN.LENGTH GREATER THAN FIELD LENGTH',
F 006 004 0042          LIMIT 0 80,
F 007 001 0043 TYPE  1  2  45  2  0  A  7  L          NOTE THE FIELD TYPE IS NUMERIC = SIGNED NUMERIC =
F 007 002 0044          SPECIAL SIGNED NUMERIC = ALPHANUMERIC OR ALPHARETIC,
F 007 003 0045          ALLOW 'N' 'SN' 'SS' 'AN' 'A' ',
F 008 001 0046 OUTPS  1  2  61  5  0  N  8  R          NOTE THE FIELD NUMBER OF THE FIELD IN OUTPUT BUFFER,
F 008 002 0047          LIMIT 0 255,
F 009 001 0048 JUST  1  2  76  1  0  A  9          NOTE RIGHT OR LEFT JUSTIFICATION IN THE FIELD,
F 009 002 0049          ALLOW 'L' 'R' ' ',
F 010 001 0050 FILL  1  3  13  1  0  AN  10          NOTE FILL CHARACTER FOR NOT USED POSITIONS IN THE FIELD,
F 010 002 0051          ALLOW ' ' '0' '*',
F 011 001 0052 REKEY  1  3  32  1  0  A  11          NOTE VERIFICATION OF THE FIELD IN REKEY=MODE,
F 011 002 0053          ALLOW 'N' 'Y' ' ',
F 012 001 0054 DISP  1  3  45  1  0  A  12          NOTE DISPLAY OF THE KEYED FIELD,
F 012 002 0055          ALLOW 'N' 'Y' ' ',
F 013 001 0056 KIND  1  3  61  1  0  A  13          NOTE THE KIND IS DUPLICATION = INCREMENT =
F 013 002 0057          CONSTANT = NOT KEYED OR KEYED,
F 013 003 0058          ALLOW 'K' 'AN' 'IC' 'D' 'I' ' ',
F 013 004 0059          IF (KIND = 'I') AND (TYPE <> 'N') THEN
F 013 005 0060          ALARM 'KIND "I" ONLY ALLOWED IF TYPE = "N"',
F 014 001 0061 REGIS  1  3  76  2  0  N  14  R          NOTE REGISTER TO HOLD DUPLICATION OR CONSTANT VALUE,
F 014 002 0062          LIMIT 1 99,

```

```

F 015 001 0063 PROG 1 5 1 80 0 AN 15 L
F 016 002 0064
F 016 003 0065
F 016 004 0066
F 016 005 0067
F 016 006 0068
F 016 007 0069
F 016 008 0070
F 016 009 0071
F 016 010 0072
F 016 011 0073
F 016 012 0074
F 016 013 0075
F 016 014 0076
F 016 015 0077
F 016 016 0078
F 016 017 0079
F 016 018 0080
F 016 019 0081
F 016 020 0082
F 016 021 0083
F 016 022 0084
F 016 023 0085
F 016 024 0086
F 016 025 0087
F 016 026 0088
F 016 027 0089
F 016 028 0090
F 016 029 0091
F 016 030 0092
F 016 031 0093
F 016 032 0094
F 016 033 0095
F 016 034 0096
F 016 035 0097
F 016 036 0098
F 016 037 0099
F 016 038 0100
F 016 039 0101
F 016 040 0102
F 016 041 0103
F 016 042 0104
F 016 043 0105
F 016 044 0106
F 016 045 0107
F 016 046 0108
F 016 047 0109
F 016 048 0110

```

```

NOTE PROGRAM STATEMENTS,
NOTE CHECK CHECKBOX CONTENTS,
COMPUTE X03 = U,
IF PAGE > 0 THEN COMPUTE X03 = X03 + 1,
IF LINE > 0 THEN COMPUTE X03 = X03 + 1,
IF POS > 0 THEN COMPUTE X03 = X03 + 1,
IF LENGTH > 0 THEN GOTO EMPTY, NOTE LENGTH = 0 OR LENGTH NOT KEYED,
NOTE 'NORMAL' = CHECKBOX,
IF (X03 < 0) AND (X03 < 3) THEN GOTO ERROR,
NOTE ERROR IN PAGENO = LINENO OR POSITION,
IF ALPHANUMERIC MINLE = X02 THEN GOTO ERROR, NOTE MINLENGTH NOT KEYED,
IF TYPE <> X02 THEN GOTO ERROR, NOTE TYPE HAS NO VALUE,
IF ALPHANUMERIC OUTPS = X02 THEN GOTO ERROR, NOTE OUTPOS NOT KEYED,
IF REGIS = 0 THEN GOTO CHE2, NOTE REGISTER NOT KEYED,
NOTE REGISTER KEYED,
IF KIND <> X02 THEN GOTO ERROR, NOTE KIND NOT KEYED,
IF KIND = 'N' THEN GOTO ERROR, NOTE KIND = NOT KEYED,
IF KIND = 'K' THEN GOTO ERROR, NOTE KIND = KEYED,
GOTO OK,
CHE2:
NOTE REGISTER NOT KEYED,
IF KIND = 'I' THEN GOTO ERROR, NOTE KIND = INCREMENT,
IF KIND = 'D' THEN GOTO ERROR, NOTE KIND = DUPLICATION,
IF KIND = 'C' THEN GOTO ERROR, NOTE KIND = CONSTANT,
GOTO OK,
NOTE 'EMPTY' / 'CONTINUATION' CHECKBOX,
EMPTY:
IF FLDNM <> X02 THEN GOTO ERROR, NOTE FIELD NAME KEYED,
IF X05 <> 0 THEN GOTO ERROR,
NOTE ERROR IN PAGENO = LINENO OR POSITION,
IF ALPHANUMERIC MINLE <> X02 THEN GOTO ERROR, NOTE MINLENGTH KEYED,
IF TYPE <> X02 THEN GOTO ERROR, NOTE TYPE KEYED,
IF ALPHANUMERIC OUTPS <> X02 THEN GOTO ERROR, NOTE OUTPOS KEYED,
IF JUST <> X02 THEN GOTO ERROR, NOTE JUSTIFICATION KEYED,
IF FILL <> X02 THEN GOTO ERROR, NOTE FILL CHARACTER KEYED,
IF REKEY <> X02 THEN GOTO ERROR, NOTE VERIFICATION KEYED,
IF DISP <> X02 THEN GOTO ERROR, NOTE DISPLAY KEYED,
IF KIND <> X02 THEN GOTO ERROR, NOTE KIND KEYED,
IF REGIS > 0 THEN GOTO ERROR, NOTE REGISTER KEYED,
GOTO OK,
NOTE ERROR IN CHECKBOX CONTENTS,
ERROR:
ALARM 'ERROR IN CHECKBOX CONTENTS',
NOTE CHECKBOX CONTENTS OK,
OK:
IF PROG = 'END SUBFORMAT,' THEN SELECT SUBFORMAT M,
IF PROG = 'END,' THEN SELECT SUBFORMAT E,
END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0111 FORM E N END FORMAT FORMAT

```

```

S FLD LIN REC NAME P LN PS LG NL TY DDI J F W D K KG PROGRAM STATEMENTS
E 001 001 0112 U END,

```

END LIST

LIST: IMAGE-SOURCE TEXT (INAU1):

```

REC FNAME S COMMENT
0001 FORM H IMAGE TO FORMAT: FORM
S REC P LN PS TEXT
H 0002 1 1 1 FORMAT NAME...:
H 0003 1 2 1 SUBFORMAT NAME:
H 0004 1 3 1 PROTECTION...:
H 0005 1 4 1 COMMENT:

```

```

REC FNAME S COMMENT
0006 FORM F
S REC P LN PS TEXT
F 0007 1 1 1 FIELD NAME:
F 0008 1 1 20 PAGE.....:
F 0009 1 1 36 LINE.....:
F 0010 1 1 49 POSITION...:
F 0011 1 2 1 LENGTH.....:
F 0012 1 2 20 MIN-LENGTH:
F 0013 1 2 36 TYPE.....:
F 0014 1 2 49 OUTPUT-POS:
F 0015 1 2 66 R/L.....:
F 0016 1 3 1 FILL.....:
F 0017 1 3 20 REKEY.....:
F 0018 1 3 36 DISPLAY:
F 0019 1 3 49 KIND.....:
F 0020 1 3 66 REGISTER:
F 0021 1 4 1 PROGRAM STATEMENTS:

```

END LIST

TRANS FORMAT WITH IMAGE STDUI IMAU1

SIZE OF IMAGE : 00370 BYTES

SIZE OF FORM : 01450 BYTES



LIST: FORMAT-SOURCE TEXT (STD02):

```
REC FNAME S P COMMENT
0001 IMAGE H N IMAGE FORMAT = SUBFORMAT HEAD ENTRY

S FLD LIN REC NAME P LN PS LG PL TY OUT J F R D K HG PROGRAM STATEMENTS

M 001 001 0002 NAME 1 1 17 5 1 AN 1 L NOTE THE FORMAT NAME IS 1 TO 5 CHARACTERS,
M 001 002 0003 IF (NAME(1)<'A') OR (NAME(1)>'Z') THEN
M 001 003 0004 ALARM 'ILLEGAL FORMAT NAME',
M 002 001 0005 SUBFW 1 2 17 1 1 AN 2 NOTE THE SUBFORMAT NAME IS 1 CHARACTER,
M 002 002 0006 IF NOT (((SUBFW='0') AND (SUBFW<'9')) OR
M 002 003 0007 ((SUBFW='A') AND (SUBFW<'Z'))) THEN
M 002 004 0008 ALARM 'ILLEGAL SUBFORMAT NAME',
M 003 001 0009 COMM 1 4 1 74 0 AN 3 L NOTE SUBFORMAT HEAD IS FINISHED -
M 004 001 0010 0 CHANGE SUBFORMAT TO TEXT DESCRIPTION,
M 004 002 0011 DEFINE X01 1, COMPUTE X01=1, NOTE LAST USED PAGE,
M 004 003 0012 SELECT SUBFORMAT F,
M 004 004 0013 END SUBFORMAT,
M 004 005 0014
```

```
REC FNAME S P COMMENT
0015 IMAGE F N TEXT DESCRIPTION ENTRY

S FLD LIN REC NAME P LN PS LG PL TY OUT J F R D K HG PROGRAM STATEMENTS

F 001 001 0016 PAGE 1 1 11 1 1 N 1 NOTE PAGENO IN DISPLAY-LAYOUT = FROM 1 TO 8,
F 001 002 0017 LIMIT 1 8,
F 001 003 0018 IF PAGE < X01 THEN
F 001 004 0019 ALARM 'CURRENT PAGENO LESS THAN PREVIOUS PAGENO',
F 001 005 0020 COMPUTE X01 = PAGE,
F 002 001 0021 LINE 1 2 11 2 1 N 2 R NOTE LINENO IN DISPLAY-LAYOUT = FROM 1 TO NO OF LINES ON DISPLAY,
F 002 002 0022 LIMIT 1 21,
F 003 001 0023 POS 1 3 11 2 1 N 3 R NOTE POSITION IN LINENO IN DISPLAY-LAYOUT = FROM 1 TO NO OF
F 003 002 0024 POSITIONS PER LINE ON DISPLAY,
F 003 003 0025 LIMIT 1 80,
F 004 001 0026 TEXT 1 5 1 80 1 AN 4 L NOTE THE DISPLAY TEXT IS 1 TO 80 CHARACTERS,
F 004 002 0027 END SUBFORMAT,
```

```
REC FNAME S P COMMENT
0028 IMAGE E N END IMAGE

S FLD LIN REC NAME P LN PS LG PL TY OUT J F R D K HG PROGRAM STATEMENTS

E 001 001 0029 0 END,

END LIST
```

LIST: IMAGE-SOURCE TEXT (IMA02):

```
REC FNAME S P COMMENT
0001 IMAGE H

S REC P LN PS TEXT

M 0002 1 1 1 FORMAT NAME...:
M 0003 1 2 1 SUBFORMAT NAME:
M 0004 1 3 1 COMMENT:
```

```
REC FNAME S P COMMENT
0005 IMAGE F

S REC P LN PS TEXT

F 0006 1 1 1 PAGE...:
F 0007 1 2 1 LINE...:
F 0008 1 3 1 POSITION:
F 0009 1 4 1 TEXT:
```

END LIST

TRANS FORMAT WITH IMAGE STD02 IMA02

SIZE OF IMAGE : 00166 BYTES

SIZE OF IMAGE : 00396 BYTES

LIST: FORMAT-SOURCE TEXT (ST003):

```
REC FNAME S P COMMENT
0001 SUBPR H N SUBPROGRAM FORMAT - HEAD

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
H 001 001 0002 NAME 1 1 18 5 1 AN 1 L IF (NAME(1)<'A') OR (NAME(1)>'Z') THEN
H 001 002 0003 ALARM 'ILLEGAL SUBPROGRAM NAME',
H 002 001 0004 COFM 1 3 1 74 0 AN 2 L
H 003 001 0005 U SELECT SUBFORMAT P,
H 003 002 0006 END SUBFORMAT,

REC FNAME S P COMMENT
0007 SUBPR P N SUBPROGRAM FORMAT - PROGRAM PARTS

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
P 001 001 0008 PROG 1 2 1 80 1 AN 1 L IF PROG = 'END,' THEN SELECT SUBFORMAT E,
P 001 002 0009 END SUBFORMAT,

REC FNAME S P COMMENT
0010 SUBPR E N SUBPROGRAM FORMAT - END

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
E 001 001 0011 U END,

END LIST
```

LIST: IMAGE-SOURCE TEXT (IMA03):

```
REC FNAME S COMMENT
0001 SUBPR H IMAGE TO FORMAT: SUBPR

S REC P LN PS TEXT
H 0002 1 1 1 SUBPROGRAM NAME:
H 0003 1 2 1 COMMENT:

REC FNAME S COMMENT
0004 SUBPR P IMAGE TO FORMAT: SUBPR

S REC P LN PS TEXT
P 0005 1 1 1 PROGRAM STATEMENTS:

END LIST
```

TRANS FORMAT WITH IMAGE ST003 IMA03

SIZE OF IMAGE : 00118 BYTES  
SIZE OF SUBPR : 00198 BYTES

LIST: FORMAT-SOURCE TEXT (ST004):

```

REC FNAME S P COMMENT
0001 TABLE 1 N TABLE FORMAT - HEAD

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
M 001 001 0002 NAME 1 1 18 5 1 AN 1 L NOTE TABLNAME IS 1 TO 5 CHARACTERS,
M 001 002 0003 IF (NAME(1)<'A') OR (NAME(1)>'Z') THEN
M 001 003 0004 ALARM 'ILLEGAL TABLE NAME',
M 002 001 0005 TYPE 1 2 18 1 1 A 2 NOTE TABLETYPE IS EITHER SINGLE DOUBLE OR MULTIPLE,
M 002 002 0006 ALLOW 'S' 'D' 'M',
M 002 003 0007 IF TYPE = 'M' THEN SKIP 4 FIELDS,
M 003 001 0008 ARG1 2 1 18 2 1 A 3 L NOTE ARGUMENTTYPE IS NUMERIC OR ALPHANUMERIC,
M 003 002 0009 ALLOW 'N' 'AN',
M 004 001 0010 ARG1 2 2 18 2 1 N 4 R U NOTE ARGUMENTLENGTH IS FROM 1 TO 80,
M 004 002 0011 LIMIT 1 80,
M 004 003 0012 IF TYPE = 'S' THEN
M 004 004 0013 SKIP 2 FIELDS,
M 005 001 0014 FUNCT 3 1 18 2 1 A 5 L NOTE FUNCTION TYPE IS NUMERIC OR ALPHANUMERIC,
M 005 002 0015 ALLOW 'N' 'AN',
M 006 001 0016 FUNCT 3 2 18 2 1 N 6 R U NOTE FUNCTION LENGTH IS FROM 1 TO 80,
M 006 002 0017 LIMIT 1 80,
M 007 001 0018 0 NOTE SELECT ACTUAL SUBFORMAT,
M 007 002 0019 IF TYPE = 'M' THEN SELECT SUBFORMAT 1,
M 007 003 0020 IF TYPE = 'D' THEN GOTO LD,
M 007 004 0021 IF ARG1 = 'N' THEN SELECT SUBFORMAT 1
M 007 005 0022 ELSE SELECT SUBFORMAT 2,
M 007 006 0023
M 007 007 0024 LD:
M 007 008 0025 IF ARG1 <> 'N' THEN GOTO LDAN,
M 007 009 0026 IF FUNCT = 'N' THEN SELECT SUBFORMAT 3
M 007 010 0027 ELSE SELECT SUBFORMAT 4,
M 007 011 0028
M 007 012 0029 LDAN:
M 007 013 0030 IF FUNCT = 'N' THEN SELECT SUBFORMAT 5
ELSE SELECT SUBFORMAT 6,
END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0031 TABLE 1 N TABLE FORMAT - SINGLE ENTRY - ARG1=N

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
1 001 001 0032 ARG 1 2 1 80 1 N 1 R U NOTE READ ARGUMENT,
1 001 002 0033 END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0034 TABLE 2 N TABLE FORMAT - SINGLE ENTRY - ARG1=AN

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
2 001 001 0035 ARG 1 2 1 80 1 AN 1 L NOTE READ ARGUMENT,
2 001 002 0036 END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0037 TABLE 3 N TABLE FORMAT - DOUBLE ENTRY - ARG1=N - FUNCT=N

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
3 001 001 0038 ARG 1 2 1 80 1 N 1 R U NOTE READ ARGUMENT,
3 002 001 0039 FUNC 1 4 1 80 1 N 2 R U NOTE READ FUNCTION,
3 002 002 0040 END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0041 TABLE 4 N TABLE FORMAT - DOUBLE ENTRY - ARG1=N - FUNCT=AN

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
4 001 001 0042 ARG 1 2 1 80 1 N 1 R U NOTE READ ARGUMENT,
4 002 001 0043 FUNC 1 4 1 80 1 AN 2 L NOTE READ FUNCTION,
4 002 002 0044 END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0045 TABLE 5 N TABLE FORMAT - DOUBLE ENTRY - ARG1=AN - FUNCT=N

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
5 001 001 0046 ARG 1 2 1 80 1 AN 1 L NOTE READ ARGUMENT,
5 002 001 0047 FUNC 1 4 1 80 1 N 2 R U NOTE READ FUNCTION,
5 002 002 0048 END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0049 TABLE 6 N TABLE FORMAT - DOUBLE ENTRY - ARG1=AN - FUNCT=AN

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
6 001 001 0050 ARG 1 2 1 80 1 AN 1 L NOTE READ ARGUMENT,
6 002 001 0051 FUNC 1 4 1 80 1 AN 2 L NOTE READ FUNCTION,
6 002 002 0052 END SUBFORMAT,

```

```

REC FNAME S P COMMENT
0053 TABLE 7 N TABLE FORMAT - DISCTABLE - NUMBER OF FUNCTIONS

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS
M 001 001 0054 0 DEFINE X01 1,
M 001 002 0055 DEFINE X02 1,
M 001 003 0056 DEFINE X03 7, MOVE '0000000' TO X03,
M 001 004 0057 DEFINE X04 1,
M 001 005 0058 DEFINE X05 1,
M 001 001 0059 NOOFF 1 1 22 1 1 N 1 NOTE NUMBER OF FUNCTIONS,
M 001 002 0060 LIMIT U 6,
M 002 001 0061 COMPUTE X01 = NOOFF,
M 002 002 0062 COMPUTE X02 = X01,
M 002 003 0063 SELECT SUBFORMAT 1,
M 002 004 0064 END SUBFORMAT,

```

REC FNAME S P COMMENT  
0005 TABLE T N TABLE FORMAT - DISCTABLE - TYPES

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS

T 001 001 0066 TYPE 1 1 31 2 1 A 1 L NOTE THE TYPE OF THE ARGUMENT OR A FUNCTION,  
T 001 002 0067 ALLOW 'AN' 'N',  
T 001 003 0068 IF TYPE = 'AN' THEN GOTO EOF,  
T 001 004 0069 IF X02 = 0 THEN  
T 001 005 0070 MOVE '1' TO X03(1)  
T 001 006 0071 ELSE  
T 001 007 0072 IF X02 = 1 THEN  
T 001 008 0073 MOVE '1' TO X03(2)  
T 001 009 0074 ELSE  
T 001 010 0075 IF X02 = 2 THEN  
T 001 011 0076 MOVE '1' TO X03(3)  
T 001 012 0077 ELSE  
T 001 013 0078 IF X02 = 3 THEN  
T 001 014 0079 MOVE '1' TO X03(4)  
T 001 015 0080 ELSE  
T 001 016 0081 IF X02 = 4 THEN  
T 001 017 0082 MOVE '1' TO X03(5)  
T 001 018 0083 ELSE  
T 001 019 0084 IF X02 = 5 THEN  
T 001 020 0085 MOVE '1' TO X03(6)  
T 001 021 0086 ELSE  
T 001 022 0087 MOVE '1' TO X03(7),  
T 001 023 0088 EOF,  
T 001 024 0089 NOTE END OF FIELD,  
T 002 001 0090 IF X02 = 0 THEN SELECT SUBFORMAT C,  
T 002 002 0091 COMPUTE X02 = X02 - 1,  
T 002 003 0092 END SUBFORMAT,

REC FNAME S P COMMENT  
0003 TABLE C N TABLE FORMAT - DISCTABLE - CONTROLWORD

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS

C 001 001 0094 CONTR 1 1 1 9 9 A 1 N NOTE A CONTROLWORD,  
C 001 002 0095 MOVE 'DISCTABLE' TO CONTR,  
C 001 003 0096 SELECT SUBFORMAT O,  
C 001 004 0097 END SUBFORMAT,

REC FNAME S P COMMENT  
0008 TABLE O N TABLE FORMAT - DISCTABLE - OPERATION

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS

O 001 001 0099 OPER 1 1 22 1 1 A 1 ALLOW 'D' 'I' 'U' 'E',  
O 001 002 0100 MOVE OPER TO X04,  
O 002 001 0101 IF OPER = 'E' THEN SELECT SUBFORMAT E,  
O 002 002 0102 COMPUTE X02 = X01 + 1,  
O 002 003 0103 SELECT SUBFORMAT S,  
O 002 004 0104 END SUBFORMAT,

REC FNAME S P COMMENT  
0105 TABLE S N TABLE FORMAT - DISCTABLE - SELECT SUBFORMAT

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS

S 001 001 0106 0 IF X02 = 0 THEN  
S 001 002 0107 SELECT SUBFORMAT O  
S 001 003 0108 ELSE  
S 001 004 0109 IF X02 = 1 THEN  
S 001 005 0110 MOVE X03(1) TO X05  
S 001 006 0111 ELSE  
S 001 007 0112 IF X02 = 2 THEN  
S 001 008 0113 MOVE X03(2) TO X05  
S 001 009 0114 ELSE  
S 001 010 0115 IF X02 = 3 THEN  
S 001 011 0116 MOVE X03(3) TO X05  
S 001 012 0117 ELSE  
S 001 013 0118 IF X02 = 4 THEN  
S 001 014 0119 MOVE X03(4) TO X05  
S 001 015 0120 ELSE  
S 001 016 0121 IF X02 = 5 THEN  
S 001 017 0122 MOVE X03(5) TO X05  
S 001 018 0123 ELSE  
S 001 019 0124 IF X02 = 6 THEN  
S 001 020 0125 MOVE X03(6) TO X05  
S 001 021 0126 ELSE  
S 001 022 0127 MOVE X03(7) TO X05,  
S 001 023 0128 IF X04 = 'D' THEN  
S 001 024 0129 COMPUTE X02 = 0  
S 001 025 0130 ELSE  
S 001 026 0131 COMPUTE X02 = X02 - 1,  
S 001 027 0132 IF X05 = 'U' THEN  
S 001 028 0133 SELECT SUBFORMAT A  
S 001 029 0134 ELSE  
S 001 030 0135 SELECT SUBFORMAT N,  
S 001 031 0136 END SUBFORMAT,

REC FNAME S P COMMENT  
0137 TABLE A N TABLE FORMAT - DISCTABLE - ALFANUMERIC ARGUMENT OR FUNCTION

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS

A 001 001 0138 ARG 1 2 1 80 1 AN 1 L SELECT SUBFORMAT S,  
A 001 002 0139 END SUBFORMAT,

REC FNAME S P COMMENT  
0140 TABLE N N TABLE FORMAT - DISCTABLE - ARGUMENT OR FUNCTION

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS

N 001 001 0141 ARG 1 2 1 80 1 N 1 0 SELECT SUBFORMAT S,  
N 001 002 0142 END SUBFORMAT,

REC FNAME S P COMMENT  
0143 TABLE E N TABLE FORMAT - END

S FLD LIN REC NAME P LN PS LG ML TY OUT J F R D K RG PROGRAM STATEMENTS

E 001 001 0144 0 END,

END LIST

LIST: IMAGE-SOURCE TEXT (IMAU4):

```
REC FNAME S COMMENT
0001 TABLE M IMAGE FOR FORMAT: TABLE

S REC P LN PS TEXT

M 0002 1 1 1 TABLE NAME.....:
M 0003 1 2 1 TABLE TYPE.....:
M 0004 2 1 1 ARGUMENT TYPE...:
M 0005 2 2 1 ARGUMENT LENGTH:
M 0006 3 1 1 FUNCTION TYPE...:
M 0007 3 2 1 FUNCTION LENGTH:

REC FNAME S COMMENT
0008 TABLE 1

S REC P LN PS TEXT

1 0009 1 1 1 ARGUMENT:

REC FNAME S COMMENT
0010 TABLE 2

S REC P LN PS TEXT

2 0011 1 1 1 ARGUMENT:

REC FNAME S COMMENT
0012 TABLE 3

S REC P LN PS TEXT

3 0013 1 1 1 ARGUMENT:
3 0014 1 3 1 FUNCTION:

REC FNAME S COMMENT
0015 TABLE 4

S REC P LN PS TEXT

4 0016 1 1 1 ARGUMENT:
4 0017 1 3 1 FUNCTION:

REC FNAME S COMMENT
0018 TABLE 5

S REC P LN PS TEXT

5 0019 1 1 1 ARGUMENT:
5 0020 1 3 1 FUNCTION:

REC FNAME S COMMENT
0021 TABLE 6

S REC P LN PS TEXT

6 0022 1 1 1 ARGUMENT:
6 0023 1 3 1 FUNCTION:

REC FNAME S COMMENT
0024 TABLE M

S REC P LN PS TEXT

M 0025 1 1 1 NUMBER OF FUNCTIONS:

REC FNAME S COMMENT
0026 TABLE T

S REC P LN PS TEXT

T 0027 1 1 1 TYPE OF ARGUMENT OR FUNCTIONS:

REC FNAME S COMMENT
0028 TABLE O

S REC P LN PS TEXT

O 0029 1 1 1 OPERATION (D,I,U,E):

REC FNAME S COMMENT
0030 TABLE A

S REC P LN PS TEXT

A 0031 1 1 1 ARGUMENT OR FUNCTIONS:

REC FNAME S COMMENT
0032 TABLE N

S REC P LN PS TEXT

N 0033 1 1 1 ARGUMENT OR FUNCTIONS:

END LIST
```

TRANS FORMAT WITH IMAGE ST004 IMAU4

SIZE OF IMAGE : 00736 BYTES

SIZE OF TABLE : 01402 BYTES



# Appendix V

## Format Language Syntax

Operand: {  
    fieldname  
    fieldname (subscripts)  
    register  
    register (subscripts)  
    constant  
}

Constant: {  
    numeric constants  
    nonnumeric constants  
}

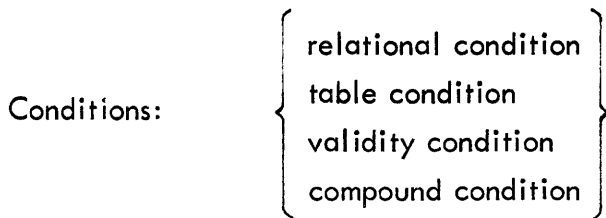
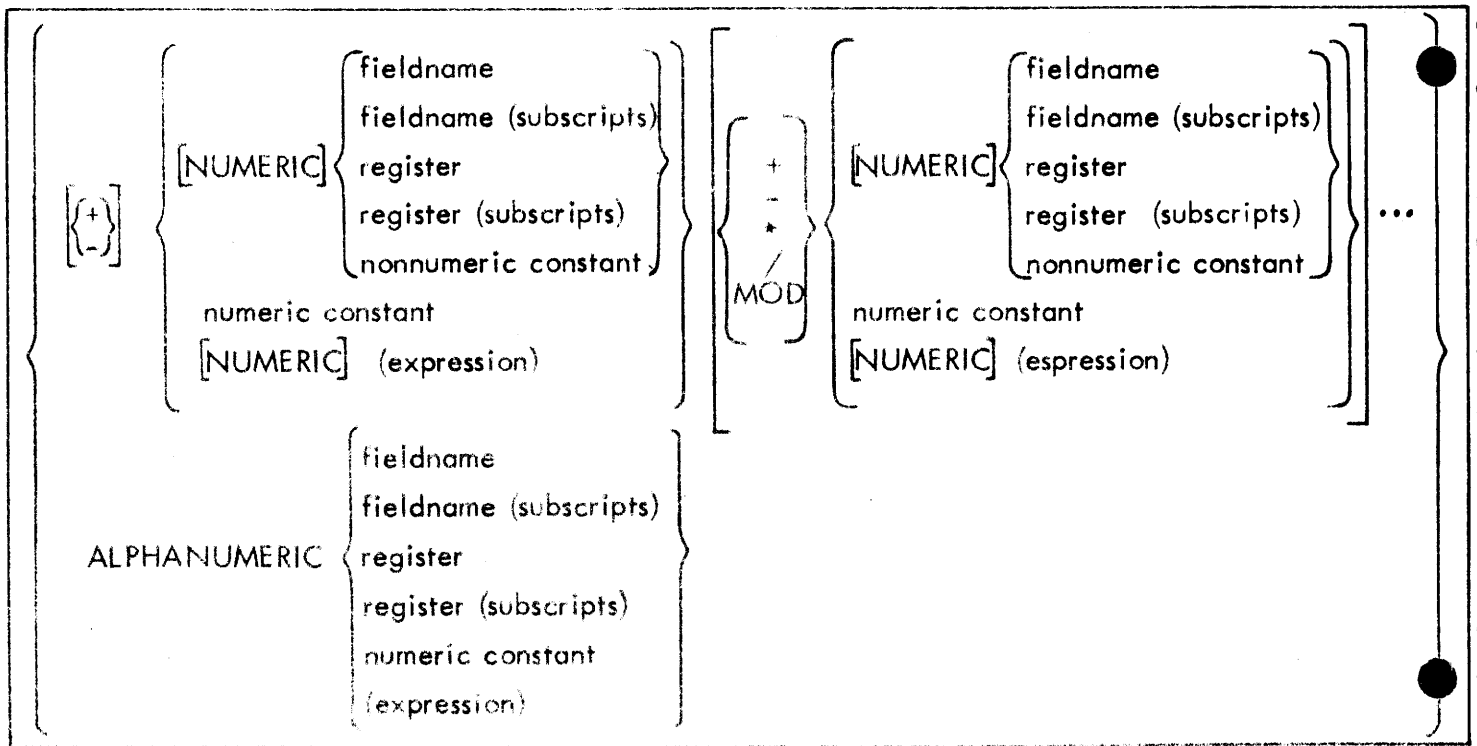
Operator: {  
    arithmetic operator  
    relational operator  
    logical operator  
}

Arithmetic operator: {  
    +  
    -  
    \*  
    /  
    MOD  
}

Relational operator: {  
    >  
    >=  
    =  
    <  
    <=  
    <>  
}

Logical operator: {  
    AND  
    OR  
    NOT  
}

Expressions:



Relational condition:

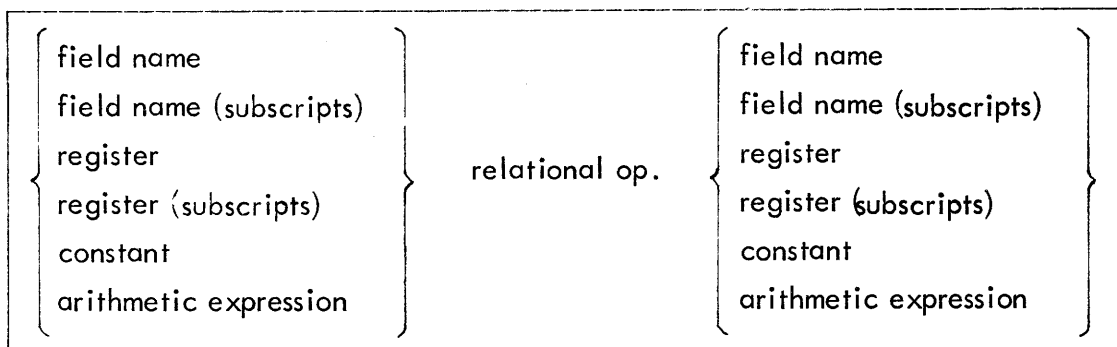
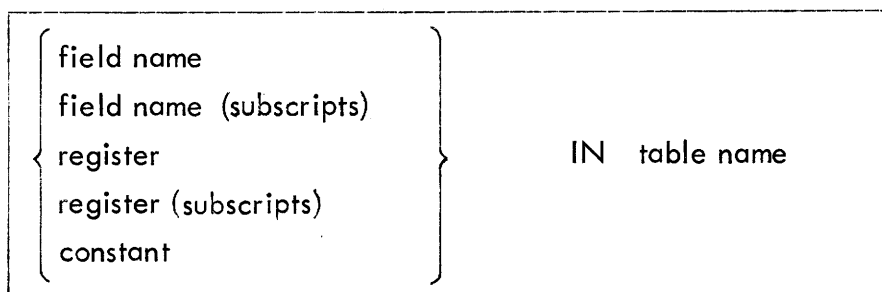
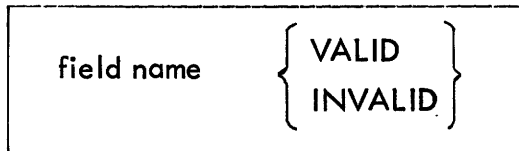


Table condition:

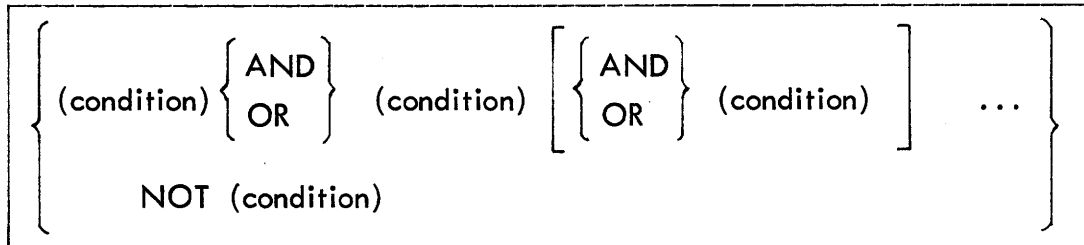




Validity condition:

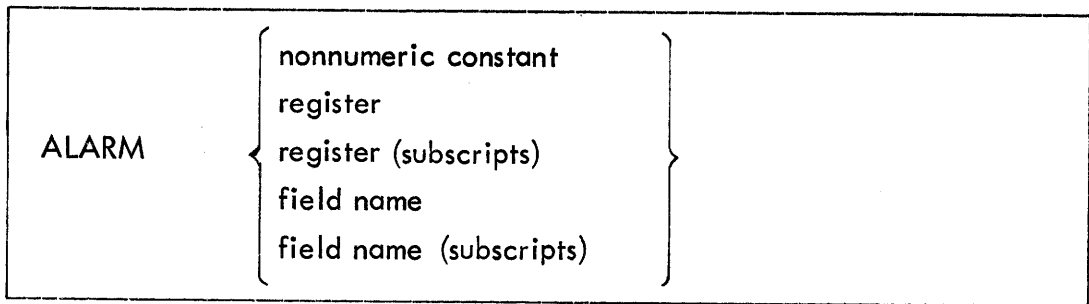


Compound condition:

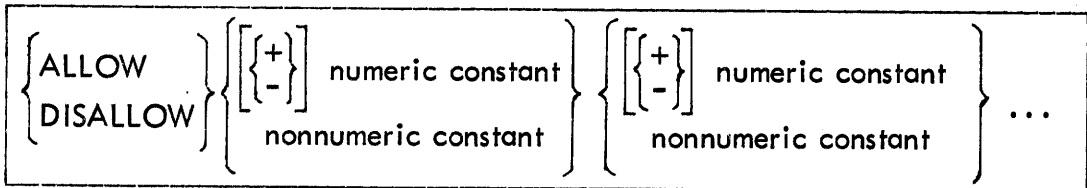


Statements:

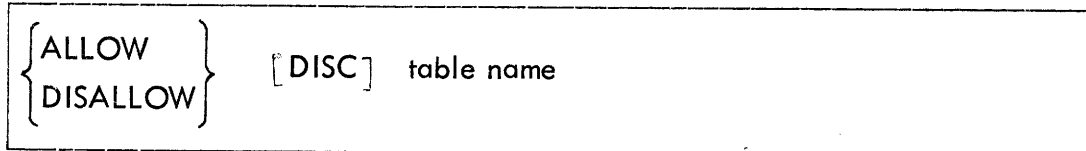
ALARM statement:



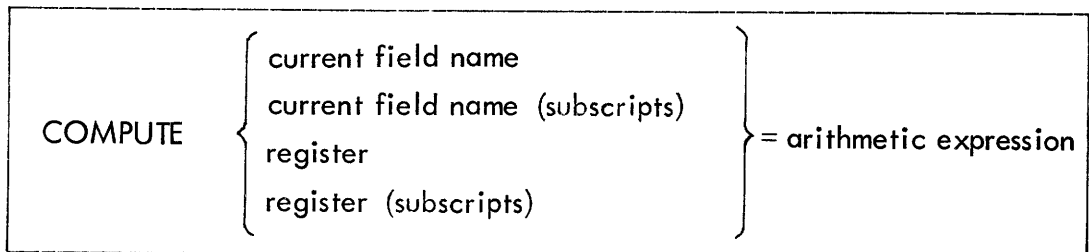
ALLOW and DISALLOW statement:



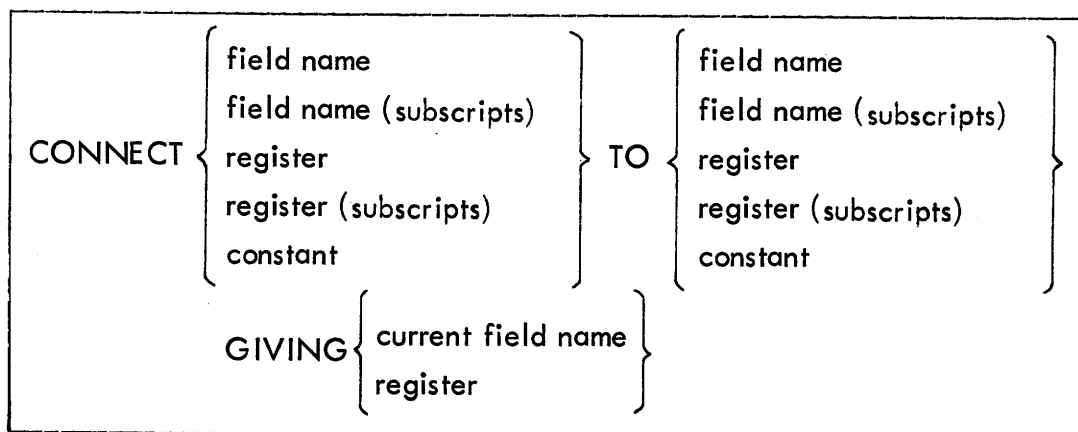
or:



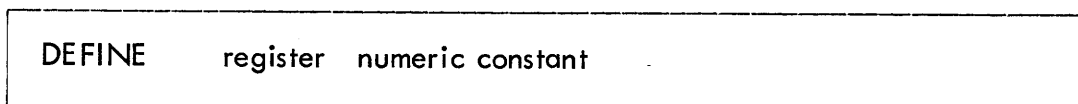
COMPUTE statement:



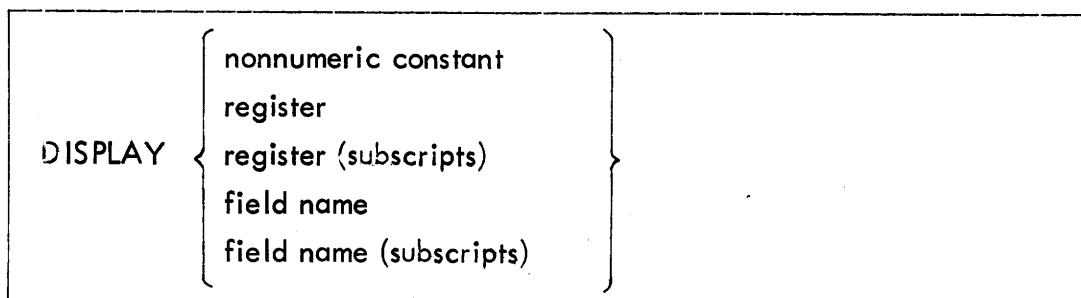
CONNECT statement:



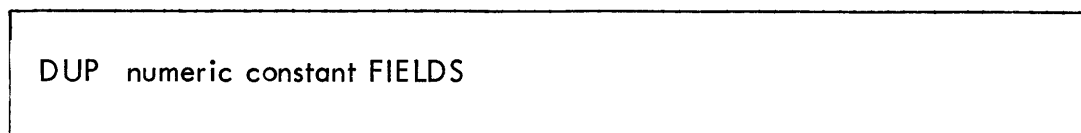
DEFINE statement:



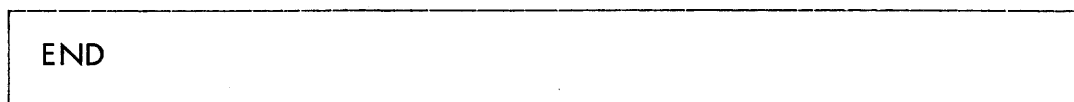
DISPLAY statement:



DUP statement:



END statement:



END SUBFORMAT statement:

END SUBFORMAT

GOTO statement:

GOTO label

LIMIT statement:

LIMIT { -numeric constant  
numeric constant } { -numeric constant  
numeric constant }

MOVE statement:

MOVE { field name  
field name (subscripts)  
nonnumeric constant  
register  
register (subscripts) } TO { current field name  
current field name (subscripts)  
register  
register (subscripts) }

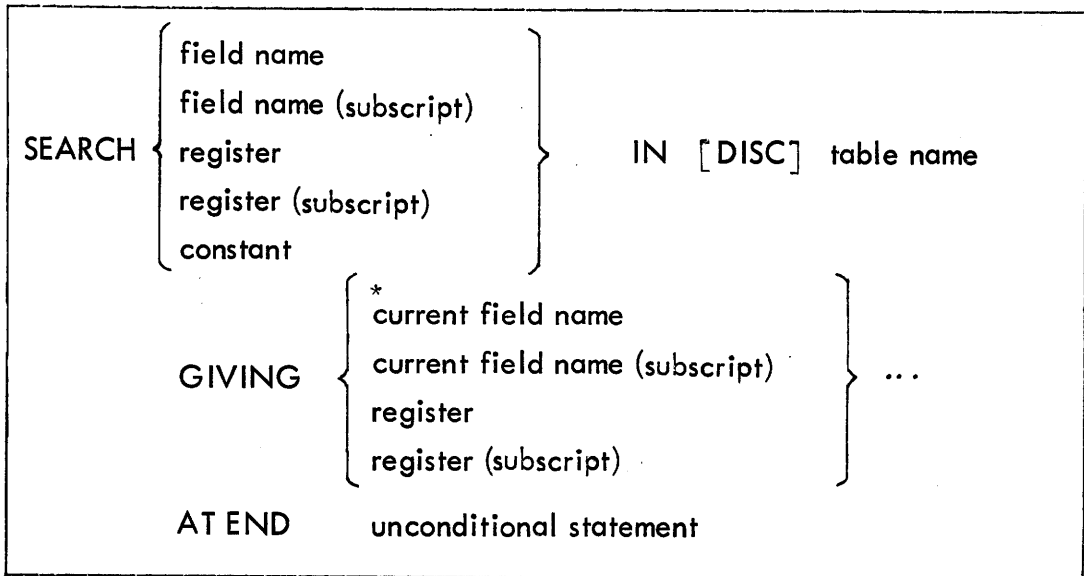
NOTE statement:

NOTE character string

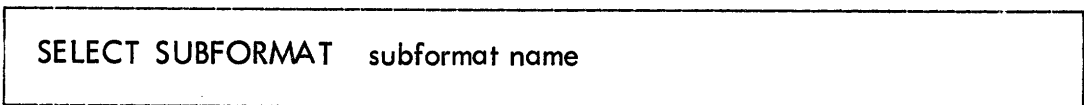
PERFORM statement:

PERFORM subprogram name

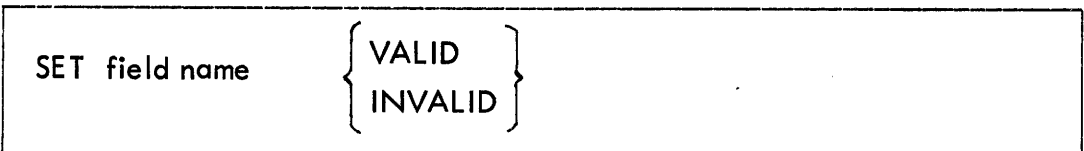
SEARCH statement:



SELECT statement:



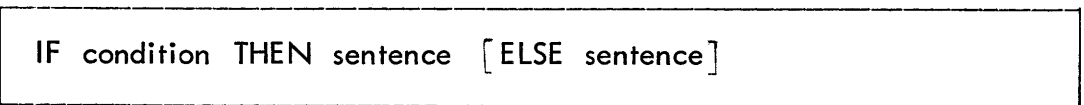
SET statement:



SKIP statement:



IF statement:





# Appendix VI

## Limitations

Maximum number of fields in a subformat .....	255
Maximum number of elements in an ALLOW/DISALLOW .....	255
Maximum number of different core tables in one TRANS call ..	10
Maximum number of different DISC tables in one TRANS call ..	20
Maximum number of different subprograms in one TRANS call ..	10
Maximum gross record length .....	20,000 bytes

Please also note that

- nonnumeric constants (i.e., '...') may not stretch over more than one line;
- no screen position may be simultaneously used both for the keying of a field and for tag specification.





# Appendix VII

## Messages From TRANSLATE

The printout produced by supervisor program TRANSLATE contains the size of the translated item or error messages if the batch to be translated contains errors.

The size is given in the following format:

SIZE OF < name >: <size.> BYTES

where <name> is the name of the format, subprogram, or table produced.

The <size> field, when a subprogram or table is translated, indicates the number of bytes (characters) occupied by the item when incorporated in a format. The <size> field given when a format is produced incorporates referenced core-tables and subprograms, but does not include an eventual image (see below) and indicates there by the (net) size of the format when loaded into memory. To calculate the gross size, add required space for registers and records.

The size of an eventual image will be given as follows:

SIZE OF IMAGE : <size > BYTES

In case of errors the printout will contain one or more of below listed messages, each message being headed by its number (as in the list).

If an error causes a stop in the execution of TRANSLATE it is noted in the following list by means of STOP; on the other hand, if an error causes a skip in the batch it is noted by SKIP. The sign - denotes that the error neither STOPS nor SKIPS.

The following cases of error printouts contain references to the batch:

- <subf> means the subformatname, which is just now handled by TRANSLATE.  
    Δ if translation of subprograms.
- <fieldno> means the fieldnumber in <subf>; counted from 1; field definitions with length = 0 or field definitions with outpos = 0 are defined by fieldnumber for latest field + relative field definition number (with length = 0 or outpos = 0).  
    0 if translation of subprograms.
- <lineno> means the linenummer relative within <fieldno>; counted from 1.
- <symbolno> means the symbolnumber relative within <lineno>; counted from 1 (e.g.:
- a field definition is one symbol;
  - ALARM, THEN, IN, etc. are one symbol each;
  - 'ONE SYMBOL' are three symbols;
  - +123 are two symbols;
  - <>, >=, <, =, etc. are one symbol each;
  - IF A <> B THEN are five symbols).

1. Illegal batchstatus <batchname> <status> STOP  
The status of the batch is invalid or not closed.
2. Illegal format <batchname> <used format> STOP  
The used format is wrong, i.e.,  
<used format> not 'FORM' in case of FORMAT-translation,  
not 'SUBPR' in case of SUBPROGRAM-translation,  
not 'TABLE' in case of TABLE-translation,  
not 'IMAGE' in case of FORMAT WITH  
IMAGE-translation.

3. Illegal subformat <batchname> <recordno> <used subformat> STOP  
 The <recordno> record in the batch has been created under control of <used subformat>, which is not allowed at this point.
- 4a. Name already exists <name> <result> STOP  
 The <name> already exists in library (i.e., format-, subprogram-, or table-library), or the library is full.  
 <result> = 1. <name> already in library,  
 = 2. library is full.
- 4b. Name already exists <batchname> <recordno> <subf> STOP  
 The <recordno> record in the batch is defining a subformat named <subf>, which has been defined already.
5. File already exists <name> <result> STOP  
 Disc error in connection with disc-file <name>  
 <result> <0 : hard error on the disc, please  
 check if disc is running,  
 = 4112 : disc-file <name> already exists,  
 = 4352 : disc-space exhausted,  
 = 4508 : fatal program error.
6. Not in use.
7. Too many fields <recordno> <subf> STOP  
 The <recordno> record in the batch is defining the 255th field in subformat <subf>.

- |      |  |        |
|------|--|--------|
| 8.   | No field <recordno> <subf><br>The <recordno> record in the batch is the first one after a subformat - head - record, and contains no checkbox (i.e., the field holding field length is empty).   | STOP   |
| 9.   | String not terminated <recordno> <subf> <fieldno><br>The program has observed a text-start-mark with no text-end-mark in the same line; the error is detected in the <recordno> record in the batch defining the <fieldno> field in subformat <subf>.  | STOP   |
| 10.  | Illegal formatname <batchname> <recordno> <formatname><br>The <recordno> record in the batch is defining a new subformat to a format identified by <formatname>.<br>The batch contains at least two subformats with unlike formatnames.  | STOP   |
| 11.  | Illegal number of records <batchname> <should have been> <was><br>The <was> record is detected to be the last one (in the logical order) in the batch, but the batch consists of <should have been> number of records.   | STOP   |
| 12.  | Not in use.  |        |
| 13.  | Double defined outpos <subf> <fieldno> <outpos> <outpos><br><outpos> output-field number has been defined twice in subformat <subf>; the error is detected at the <fieldno> field.   | SKIP   |
| 14a. | Illegal statement type <subf> <fieldno><br>An end-statement is written in the last field in subformat <subf>, but this subformat is not the last one in the format, no skip.<br>An end-, endsubformat-, or select-statement is detected in <fieldno> field in subformat <subf>, but this field is not the last one in the subformat, skip. | -/SKIP |

- 14b. Illegal statement type -  
 The program has found a set-statement, an endsubformat-statement, or a select-statement in a subprogram, or the end-statement is missing.
- 14c. Illegal statement type <subf> <fieldno> <lineno> <symbolno> SKIP  
 TRANSLATE has detected an illegal symbol after an end-statement or an endsubformat-statement; these statements are to be followed by comma and nothing else.
15. Fatal program error <where> STOP  
 Fill in an error report for Regnecentralen.  
 <where> = 0 : X01 does not exist as a symbol in symtab,  
 = 1 : tpass does not contain an end-subformat-mark after last field in a subformat,  
 = 2 : tpass contains more subformats than counted in subformat head (observed in connection with new subformat),  
 = 3 : tpass does not contain an end-format-mark after last subformat,  
 = 4 : tpass contains the wrong number of subformats (observed in connection with end format).
16. Subformat not terminated <subf> -  
 Subformat <subf> contains no end-statement and no end-subformat-statement.
17. Illegal number of arguments <should have been> <was> STOP  
 The program has detected the <was> argument to be the last one (in the logical order), but it was expecting <should have been> number of arguments.
18. Format not terminated -  
 End-statement is missing in last field in last subformat.

19. Insert error <name> <result> -  
 An error occurs inserting <name> in library (i.e., format-, subprogram-, or table-library).  
 <results> = 1 : <name> already in library,  
 = 2 : library is full.
20. Too many tables <subf> <fieldno> <lineno> <symbolno> STOP  
 There are references to too many (different) tables. Concerning parameters, see above.
21. Double defined <subf> <fieldno> <lineno> <symbolno> STOP/SKIP  
 Two (or more) identifiers with the same name. Concerning parameters, see above.
22. Illegal symbol <subf> <fieldno> <lineno> <symbolno> SKIP  
 Last read symbol not allowed at this point (normally, syntax error). Concerning parameters, see above.
23. Illegal terminator <subf> <fieldno> <lineno> <symbolno> SKIP  
 Statement not terminated by comma or label definition not terminated by colon. Concerning parameters, see above.
24. Undefined <subf> <fieldno> <lineno> <symbolno> SKIP  
 Item is not defined or does not exist, e.g., table/subprogram not in library.  
 Concerning parameters, see above.
25. Illegal type <subf> <fieldno> <lineno> <symbolno> SKIP  
 Item has been detected to be with an invalid type, e.g.,  
 - a keyed field has been used as destination (e.g., in a compute-statement),  
 - expression between if and then is not a relation,  
 - nonnumeric fields (or constants) occur in arithmetic expression.  
 Concerning parameters, see above.
26. Stack <subf> <fieldno> <lineno> <symbolno> STOP  
 There is no room for creating current format, the causes may be

- too many fields in one subformat,
- expression with a structure too complicated (e.g., many brackets).

You can paraphrase the format to a simpler one (i.e., a format, where all expressions are dispersed into simple ones and all fields (not referred) are identified by field-name consisting of 5 spaces). If the paraphrasing has no effect, please fill in an error report for Regnecentralen. Concerning parameters, see above.

- |     |   |      |
|-----|---|------|
| 27. | Illegal expression <subf> <fieldno> <lineno> <symbolno><br>The program has observed an expression before 'IN' in a table condition.<br>Concerning parameters, see above.  | SKIP |
| 28. | Illegal registerno <subf> <fieldno> <lineno> <symbolno><br>The program has found a reference to register zero.<br>Concerning parameters, see above.   | SKIP |
| 29. | Illegal index value <subf> <fieldno> <lineno> <symbolno><br>A subscript has to be greater than zero and less than 256.<br>Concerning parameters, see above.   | SKIP |
| 30. | Too many items <subf> <fieldno> <lineno> <symbolno><br>It is not permitted to have more than 255 strings in an allow- or disallow-statement, to define a register with more than 255 characters or to skip more than 255 fields.<br>Concerning parameters, see above. | SKIP |
| 31. | Illegal recstatus <batchname> <recordno> <recstatus><br>The <recordno> record contains an input error.  | STOP |
| 32. | Illegal format structure <batchname><br>The batch is not terminated by a record created by sub-format E.  | STOP |
| 33. | Not in use.   | STOP |

34. Remove error <name> <result> -  
 Disc error in connection with disc-file <name>  
 <result> < 0 : hard error on the disc, please  
 check if disc is running,  
 = 20480: disc-file <name> does not exist,  
 = 4508: fatal program error, or  
 disc-file <name> used by an-  
 other user.
- 35a. Illegal length <recordno> <entryno> -  
 An argument or a function to a table-definition has a  
 wrong length (normally too long).  
 The error has been detected in the <recordno> record  
 in the batch, this record is defining the <entryno>  
 entry in the table.
- 35b. Illegal length <subf> <recordlength> -  
 The resulting <recordlength> from <subf> is greater than  
 than 20,000.
36. Illegal outpos <subf><fieldno> <outpos> SKIP  
 The checkbox for the <fieldno> field in subformat  
 <subf> is defining an output-position <outpos> greater  
 than number of fields.
37. Undefined label <subf> <fieldno> <lineno> <symbolno> -  
 The program has not found a label definition for a label  
 referred.  
 Concerning parameters, see above.
- 38a. No room in current line <subf> <fieldno> <lineno> -  
 <symbolno>  
 There is not enough space in current screen line for  
 current field.  
 Concerning parameters, see above.



- 38b. No room in current line <batchname> <subf> <page> -  
<line> <position>  
There is not enough space in current screen line for the  
image text given by <subf>, <page>, <line> and  
<position>.
- 39a. Line too large <subf> <fieldno> <lineno> <symbolno> -  
<line> (field description, column 3) is greater than num-  
ber of datalines in the screen.  
Concerning parameters, see above.
- 39b. Line too large <batchname> <subf> <page> <line> -  
<position>  
<line> is greater than number of datalines in the screen.  
The parameters describe current image text.
- 40a. Current page less than previous page <subf> <fieldno> -  
<lineno> <symbolno>  
The page numbers must occur in a not descending order  
inside the subformat.  
Concerning parameters, see above.
- 40b. Current page less than previous page <batchname> <subf> -  
<page> <line> <position>  
The page numbers must occur in a not descending order  
inside the image for one subformat.  
The parameters describe current image text.
- 41a. Page too large <subf> <fieldno> <lineno> <symbolno> -  
<page> (field description, column 2) is greater than 8.  
Concerning parameters, see above.
- 41b. Page too large <batchname><subf> <page> <line> -  
<position>  
<page> is greater than 8.  
The parameters describe current image text.

- 42a. Screen position used more than once <subf> <fieldno> -  
 <lineno> <symbolno>  
 At least one of the screen positions for current field has  
 been reserved by a previous field.  
 Concerning parameters, see above.
- 42b. Screen position used more than once <batchname> <subf> -  
 <page> <line> <position>  
 At least one of the screen positions for current text has  
 been reserved by a previous image text.  
 The parameters describe current image text.
43. Subformat does not exist <batchname> <recordno> <subf> STOP  
 The <subf> referred in the <recordno> record of the  
 image-batch <batchname> has not been defined in the  
 format.
44. Screen position used both by tag and by field -  
 <subf> <page> <line> <from pos> <to pos>  
 At least one of the screen positions in the interval <from pos>  
 to <to pos> is used both by a tag and by a field. The po-  
 sitions are allocated to <line> line in the <page>  
 page of subformat <subf>.
45. Disc tables not allowed in subprograms <subf> <fieldno> SKIP  
 <lineno> <symbolno>  
 References to DISC tables must not occur in subprograms.  
 Concerning parameters, see above.
46. Registername <> tablename <subf> <fieldno> <lineno> SKIP  
 <symbolno> <registername>  
 The <registername> stored in the disc-table pointed out  
 by the first four parameters does not equal the tablename.  
 Concerning parameters, see above.
47. Too many subprograms <subf> <fieldno> <lineno> <symbolno> STOP  
 There are references to too many (different) tables.  
 Concerning parameters, see above.

In case of disc trouble not described above TRANS prints

<name> error <code>

and aborts its execution. <name> defines the disc-file in question, and <code> is a disc error code, see Users Guide Part 2, Appendix 2.

When TRANSLATE has finished, it will produce a receipt, see Users Guide Part 2, Sections 8 and 9.



# Appendix VIII

## Definitions of Terms

For explanation in full of terms used in the RC 3600 Data Entry System, see the following sections:

'	3.2, 3.2.5
Argument	6.6
Arithmetic expressions	3.5
Arithmetic operators	3.2.2
Batch	1.1.1
- translation	2
Character set	3.2
Compound conditions	3.6
Conditional statement	3.7
Constant	3.1.3
Conversion operator	3.5
Disc file	1.1.1
Field	1.1.3
Field flag	4.5
Field program	3.1.3
Fill-in-the-blanks	1.5.1
Format	1.2.1
Format program	1.2.1
Function	6.6
Image	1.5
Input parameter	3.1.1
Logical operators	3.2.4
Multiplying operator	3.5
Nonnumeric operands	3.3
Numeric operands	3.3
Operands	3.3
Operators	3.2.2, 3.2.3, 3.2.4
Output parameter	3.1.1
Program	3.1.1

Record	1.1.2
Register	3.3.2
Relation	3.6.1
Relational operator	3.2.3
Statements	3.7
Subformat	1.2.2
Subformat image	1.5.2
Subprogram	1.3.1
Subscripts	3.3.4
Table	6.6
Tag	1.5.1
Unconditional state- ments	3.7
Variable	3.1.3

# Appendix IX

## Index

Numbers referring to figures are underlined>.

...	3.4
[ ]	3.4
{,}	3.4
△	3.2, 3.2.5
+	3.2, 3.2.2, 3.5
-	3.2, 3.2.2, 3.5
*	3.2, 3.2.2, 3.5
/	3.2, 3.2.2, 3.5
=	3.2, 3.2.3, 3.6.1
,	3.2, 3.2.5
;	3.2, 3.2.5
'	3.2, 3.2.5
(	3.2, 3.2.5, 3.5
)	3.2, 3.2.5, 3.5
>	3.2, 3.2.3, 3.6.1
>=	3.2.3, 3.6.1
<	3.2, 3.2.3, 3.6.1
<=	3.2.3, 3.6.1
:	3.2, 3.2.5, 3.7.1. 10
<>	3.2.3, 3.6.1
A-length	2.3.1.1, 2.3.2.1
A-type	2.3.1.1, 2.3.2.1
Addition	3.2.2
Adding operators	3.5
ALARM	3.2.1, 3.7.1.1, 3.8.1
ALLOW	3.2.1, 3.7.1.2, 3.8.1
ALPHANUMERIC	3.2.1,3.2.2,3.5,3.6.1
AND	3.2.1, 3.2.4, 3.6.4
Argument	1.3.3, 2.3.1, 2.3.2,2.3.3,6.6
- description	2.3.1, 2.3.2 , 2.3.3
Arithmetic expressions	3.5
Arithmetic operators	3.2.2

Asterisk (*)	3.2, 3.2.2, 3.5, 3.7.1.14
AT	3.2.1, 3.7.1.15
Automatic duplication	6.4
Automatic fields	4.4.2
Automatic incrementation	6.5
Automatic insertion	6.3
Batch	1.1.1
- translation	2
BYPASS	4.4.7
Character set	3.2
Coding sheets	2
Colon (:)	3.2
Comma (,)	3.2
Comment	2.1.1.1, 2.1.2.1, 2.2.1, 3.2.1, 3.7.1.13, 3.8.1
Compound conditions	3.6
COMPUTE	3.2.1, 3.7.1.3, 3.8.1
Condition	
- compound	3.6
- relation	3.6.1
- simple	3.6
- table	3.6.2
- validity	3.6.3
Conditional statement	3.7, 3.7.2
CONNECT	3.2.1, 3.7.1.4, 3.8.1
Constant	3.1.3, 3.3.1
Constant field	4.4.2.2, 6.3.2
Conversion operators	5.3, 3.6.1
Core table	5.3.2
DEFINE	3.2.1, 3.7.1.5, 3.8.1
Defining tags	6.1.3
Definitions of terms	A VIII
Digit	3.2, 3.2.1, 3.3.1, 3.3.2
DISALLOW	3.2.1, 3.7.1.2, 3.8.1



Disc	
- file	1.1.1
- required space	A II
- table	5.3.2, 6.6
DISC	3.2.1, 3.7.1.2, 3.7.1.15
DISCTABLE	5.3.2
Display	2.1.1.2, 3.2.1, 3.7.1.6, 3.8.1
Division	3.2.2, 3.5
Double entry table coding sheet	2.3.2, <u>2.3.2</u>
DUP	3.7.1.7
Duplicate field	4.4.2.1, 6.4
EDIT	4.5.4
ELSE	3.2.1, 3.7.2.1
END	3.2.1, 3.7.1.8, 3.7.1.15, 3.8.1
END SUBFORMAT	3.2.1, 3.7.1.9
ENTER	4.4.5
Equal to (=)	3.2, 3.2.3, 3.6.1
Error messages	
- ALARM	3.7.1.1
- from TRANSLATE	A VII
Examples	A III
Expressions, arithmetic	3.5
F-length	2.3.2.1
F-type	2.3.2.1
Fake	3.6
Field	1.1.3, 3.3.3
- alphabetic	2.1.1.2
- alphanumeric	2.1.1.2
- automatic	4.4.2
- constant	4.4.2.2, 6.3.2
- definition	1.2.4, 3.3.3
- duplicate	4.4.2.1
- fill characters	2.1.1.2
- increment	4.4.2.3
- keyed	4.4.1
- kind	2.1.1.2
- name	2.1.1.2
- not keyed	4.4.3, 6.3.1

Field (contd.)	
- numeric	2.1.1.2
- output position	2.1.1.2
- right/left justification	2.1.1.2
- signed numeric	2.1.1.2
- special signed numeric	2.1.1.2
FIELD	3.2.1, 3.7.1.18
Field description	1.2.3, 2.1.1.2
- execution	4.4
Field flag	4.5
- for EDIT	4.5.4
- for REKEY	4.5.3
- skipped	4.5.2
- validity	4.5.1
Field program	1.2.4, 3.1.3
- execution	4.4.8
Fill characters	2.1.1.2
Fill-in-the-blanks	1.5.1
- mask	1.5.3
Flag, validity	4.5.1
FORM standard format	5.1, A IV
Format	1.2.1
- coding sheet	2.1.1, <u>2.1.1</u>
- image	1.5.2, 4.8
- name	2.1.1.1, 2.1.2.1
- new formats	5.1
Format language	3.1.3
- examples	3.1.4, A III
- statements	3.7
- syntax	A V
Format program	1.2.1, 3, 4, 6
- execution	4
- standard	A IV
Function	1.3.3, 2.3.2, 2.3.3, 6.6
- description	2.3.2, 2.3.3
- length	2.3.2.1
- type	2.3.2.1, 2.3.3.1

GIVING	3.2.1, 3.7.1.4, 3.7.1.15
GOTO	3.2.1, 3.7.1.10, 3.8.1
Greater than (>)	3.2, 3.6.1
Greater than or equal to (>=)	3.2.3, 3.6.1
IF	3.2.1, 3.7.2.1
Image	1.5
- coding sheet	2.1.2, <u>2.1.2</u>
- format image	1.5.2, 2.1.2
- page	1.5.3, 2.1.2.2
- subformat image	1.5.2, 2.1.2.1
- text	2.1.2.2
IMAGE, standard format	5.1, A IV
IN	3.2.1, 3.7.1.14
Increment field	4.4.2.3, 6.5
Input parameter	3.1.1
INVALID	3.2.1, 3.6.3, 3.7.1. 17, 4.5.1
Invoice	1.1.3, <u>1.1.3</u>
Keyed field	4.4.1
Keying positions	6.1.2
Kind	2.1.1.2
Left parenthesis (( )	3.2, 3.5
Length	2.1.1.2
- minimum	2.1.1.2
Less than (<)	3.2, 3.6.1
Less than or equal to (<=)	3.2.3, 3.6.1
Letter	3.2, 3.2.1
LIMIT	3.2.1, 3.7.1.11, 3.8.1
Limitations	A VI
Line	2.1.1.2, 2.1.2.2
Logical operators	3.2.4
Minimum length	2.1.1.2
Minus (-)	3.2, 3.2.2, 3.5

MOD	3.2.1, 3.2.2, 3.5
Modulo	3.2.2
MOVE	3.2.1, 3.7.1.12, 3.8.1
Multiplication	3.2.2
Multiplying operators	3.5
Names, reserved	3.2.1
New formats	5.1
New subprograms	5.2
New tables	5.3
Nonnumeric operands	3.3, 3.6.1.2
NOT	3.2.1, 3.2.4, 3.6.4
Not equal to (<>)	3.2.3, 3.6.1
Not keyed fields	4.4.3, 6.3.1
- skipped by BYPASS	4.4.7
- skipped by ENTER	4.4.5
- skipped by RECORD RELEASE	4.4.6
- skipped by SKIP	4.4.4
Notation	3.4
NOTE	3.2.1, 3.7.1.13, 3.8.1
NUMERIC	3.2.1, 3.2.2, 3.5, 3.6.1, 3.7.1.3
Numeric operands	3.3, 3.6.1.1
Operands	3.3
- in subprograms	3.8.2
- nonnumeric	3.3
- numeric	3.3
Operators	3.2.2, 3.2.3, 3.2.4
- arithmetic	3.2.2, 3.5
- logical	3.2.4, 3.6.4
- relational	3.2.3, 3.6.1
OR	3.2.1, 3.2.4, 3.6.4
Output parameter	3.1.1
Output position	2.1.1.2
Page	2.1.1.2, 2.1.2.2
Parameter	3.1.1
Partial rekeying	6.7

PERFORM	3.2.1, 3.7.1.14, 3.8.1
Plus (+)	3.2, 3.2.2, 3.5
Position	2.1.1.2, 2.1.2.2
Program	3.1.1, 6
- elements	3.1.2
- execution	4
- planning	3.1.4
- statements	2.1.1.2, 2.2.2
Programming hints	6
Protected	2.1.1.1
Pseudo-register	1.4, 3.3.2, 6.8
Punctuation symbols	3.2.5
Quotation (')	3.2, 3.3.1
Record	1.1.2
RECORD RELEASE	4.4.6
Reformatting	6.2
Register	1.4, 2.1.1.2, 3.3.2, 4.6
REKEY	2.1.1.2, 4.5.3
Relation	3.6.1
Relational operators	3.2.3
Replay	4.7
Required space	
- in translated format A I	
- on disc for batches A II	
Reserved names	3.2.1
Reserved verbs	3.2.1
Right/left justification	2.1.1.2
Right parenthesis ( )	3.2, 3.5
Screen processing	6.1
- assigned to system	6.1.1
SEARCH	3.2.1, 3.7.1.15, 3.8.1
SELECT	3.2.1, 3.7.1.16
SELECT SUBFORMAT	3.2.1
Semicolon (;)	3.2
SET	3.2.1, 3.7.1.17

Single entry table coding sheet	2.3.1, 2.3.1
SKIP	3.2.1, 3.7.1.18, 3.8.1, 4.4.4
Skipped flag	4.5.2
Skipped not keyed field	
- by BYPASS	4.4.7
- by ENTER	4.4.5
- by RECORD RELEASE	4.4.6
- by SKIP	4.4.4
Space (A)	3.2
Space, required	A I, A II
Standard formats	A IV
Statements	3.7
- conditional	3.7.2
- in subprograms	3.8.1
- unconditional	3.7.1
Stroke (/)	3.2, 3.2.2, 3.5
Subformat	1.2.2
- execution	4.3
- head	2.1.1.1, 2.1.2.1
- name	2.1.1.1, 2.1.2.1
- selection	4.1
- termination	4.2
SUBFORMAT	3.2.1, 3.7.1.16
Subformat image	1.5.2
SUBPR standard format	5.2, A IV
Subprogram	1.3.1, 3.1.3, 3.8
- coding sheet	2.2, 2.2
- head	2.2.1
- name	2.2.1
- operands in	3.8.2
- statements in	3.8.1
Subscripts	3.3.4
Subtraction (-)	3.2.2

Table	1.3.2, 6.6
- coding sheet, single entry	2.3.1, <u>2.3.1</u>
- coding sheet disc table	2.3.3, <u>2.3.3</u>
- coding sheet, double entry	2.3.2, <u>2.3.2</u>
- condition	3.6.2
- core table	5.3.1
- DISC table	2.3.3, 5.3.1, 5.3.2, 6.6
- head, single entry	2.3.1.1
- head, disc table	2.3.3.1
- head, double entry	2.3.2.1
- name	2.3.1.1
- the use of	6.6
- type	2.3.1.1, 2.3.2.1, 2.3.3.1
TABLE standard format	5.3, A IV
Tag	1.5.1
- defining tags	6.1.3
- description	2.1.2
Text	2.1.2.2
THEN	3.2.1, 3.7.2.1
TO	3.2.1, 3.7.1.4, 3.7.1.12
Tone	3.6
TRANSLATE	5, A VII
VALID	3.2.1, 3.6.3, 3.7.1.17, 4.5.1
Validity	3.6.3
- condition	3.6.3
- flag	4.5.1
Variable	3.1.3
XBATCH	3.2.1, 3.3.2
XDATE	3.2.1, 3.3.2
XJOB	3.2.1, 2.2.2
XOPERATOR	3.2.1, 3.3.2
XTIME	3.2.1, 3.3.2





NOTES

NOTES



NOTES

NOTES



READER'S COMMENTS

A/S Regnecentralen maintains a continuous effort to improve the quality and usefulness of its publications. To do this effectively we need user feedback - your critical evaluation of this manual.

Please comment on this manual's completeness, accuracy, organization, usability, and readability:

---

---

---

Do you find errors in this manual? If so, specify by page.

---

---

---

---

How can this manual be improved?

---

---

---

---

Other comments?

---

---

---

---

---

Please state your position: \_\_\_\_\_

Name: \_\_\_\_\_ Organization: \_\_\_\_\_

Address: \_\_\_\_\_ Department: \_\_\_\_\_

\_\_\_\_\_  
\_\_\_\_\_  
Date: \_\_\_\_\_

Thank you!

----- Fold here -----

----- Do not tear - Fold here and staple -----

Affix  
postage  
here

---

A/S REGNECENTRALEN  
Marketing Department  
Falkoner Allé 1  
2000 Copenhagen F  
Denmark



## INTERNATIONAL

### **EASTERN EUROPE**

A/S REGNECENTRALEN  
Glostrup, Denmark, (02) 96 53 66

## SUBSIDIARIES

### **AUSTRIA**

RC – SCANIPS COMPUTER  
HANDELSGESELLSCHAFT mbH  
Vienna, (0222) 36 21 41

### **FINLAND**

OY RC – SCANIPS AB  
Helsinki, (90) 31 64 00

### **FRANCE**

RC – COMPUTER S.a.r.l.  
Paris

### **HOLLAND**

REGNECENTRALEN (NEDERLAND) B.V.  
Rotterdam, (010) 21 62 44

### **NORWAY**

A/S RC – SCANIPS  
Oslo, (02) 35 75 80

### **SWEDEN**

RC – SCANIPS AB  
Stockholm, (08) 34 91 55

### **SWITZERLAND**

RC – SCANIPS (SCHWEIZ) AG  
Basel, (061) 22 90 71

### **UNITED KINGDOM**

REGNECENTRALEN LTD.  
London, (01) 439 93 46

### **WEST GERMANY**

RC – GIER ELECTRONICS G.m.b.H.  
Hannover, (0511) 67 971

RC – COMPUTER G.m.b.H.  
Hannover, (0511) 63 99 51

## REPRESENTATIVES

### **HUNGARY**

HUNGAGENT AG  
Budapest, 88 61 80

### **KUWAIT**

KUWAIT – DANISH COMPUTER CO. S.A.K.  
Kuwait, 51 05 10

### **CZECHOSLOVAKIA**

KSNP KANCELARSKE STROJE N.P.  
Praha, 27 00 01

## TECHNICAL ADVISORY REPRESENTATIVES

### **POLAND**

ZETO  
Wroclaw, 44 54 31

### **RUMANIA**

I.I.R.U.C.  
Bucharest, 33 21 57

### **HUNGARY**

NOTO-OSZV  
Budapest, 66 84 11

**RC** REGNECENTRALEN  
**Scanips**  
**COMPUTER**

HEADQUARTERS: FALKONER ALLE 1 · DK-2000 COPENHAGEN F · DENMARK  
PHONE: (01) 10 53 66 · TELEX: 162 82 rc hq dk · CABLES: REGNECENTRALEN