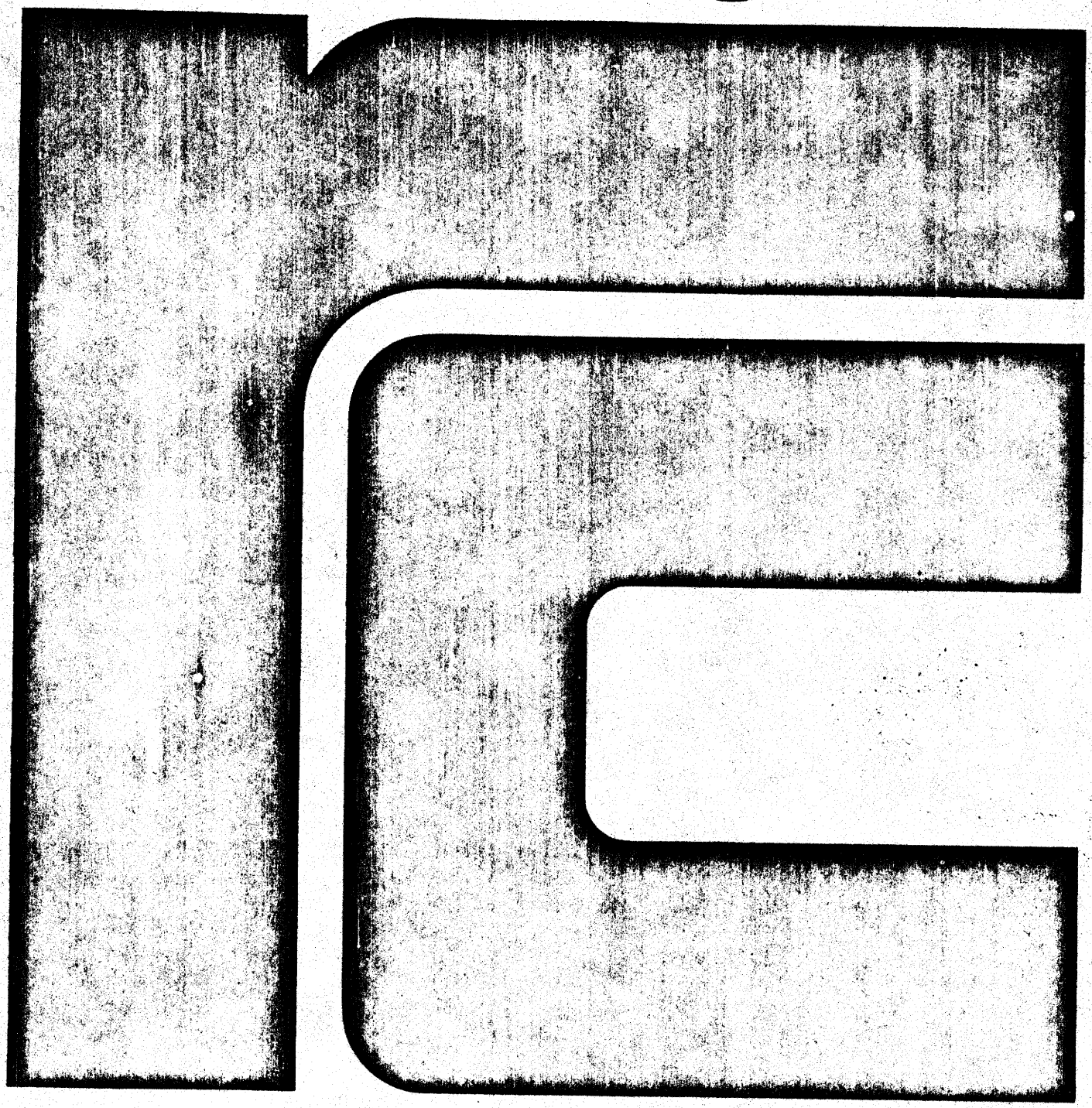
DATA ENTRY
Release 2

Format Language Guide



3600

RC 3600 Data Entry
Format Language Guide

Release 2

A/S REGNECENTRALEN
Marketing Department

First Edition
September 1976
RCSL 42 - i 0395

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

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.

Printed by A/S Regnecentralen, Copenhagen

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

The RC 3600 Data Entry System is a software package operating under the RC 3600 Multiprogramming Utility System.

It is an input data preparation key-to-disc system, receiving data from local or remote CRT 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 Manual 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.

FRUIT MARKET Inc.

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

Customer/Kunde:	Delivered to/Leveret til:

Invoice date/Faktura dato:	Terms of payment/Betalingsbet.:	Date shipped/Forsend. dato:	Shipped by/Sendt med:
Your ref./Deres ref.:	Our ref./Vor ref.:	Shipped from/Sendt fra:	Shipped to/Sendt til:
Customers no./Kunde nr. / 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:		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:

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 double-entried.

The 1st column is called argument;
The 2nd column is called function.

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 a register can always be changed by a field program.

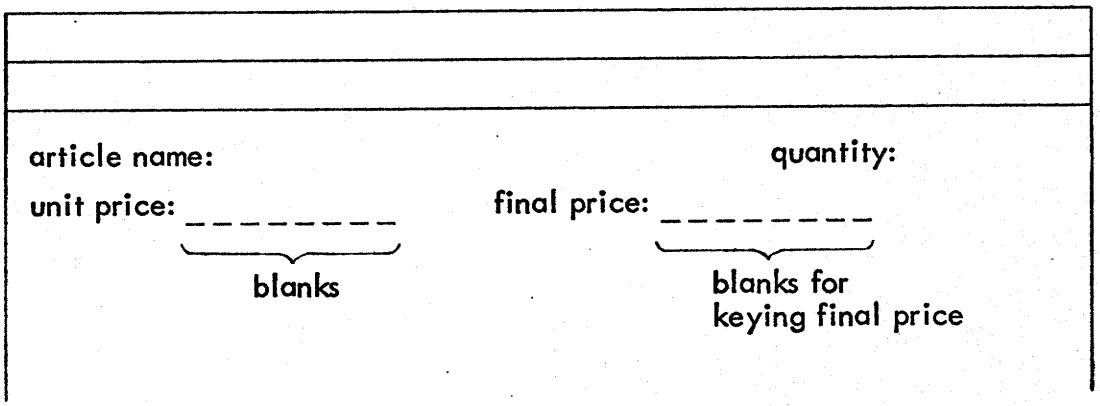
Registers may be of different length.

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.



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 TRANS 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.

C3600 Data Entry System

FORMAT CODING SHEET

INITIALS:

DATE:

NOTES:

PAGE OF

FORMAT:

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	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 = , △, *, 0 – 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.

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 (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 (col. 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 (col. 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.

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

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

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.

C3600 Data Entry System	IMAGE CODING SHEET		NOTES:	PAGE OF
	INITIALS:	DATE:		FORMAT:

FIGURE 2.1.2

FORMAT NAME	S	COMMENT
1	2	3

PAGE	LINE	POSITION	TEXT
1	2	3	4

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

The subprogram coding sheet consists of two parts:

subprogram head, and
subprogram parts.

See Figure 2.2.

2.2.1 Subprogram Head

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

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!)

C3600 Data Entry System	**SUBPROGRAM CODING SHEET**		NOTES:	PAGE OF
	INITIALS:	DATE:		SUBPROGRAM:

FIGURE 2.2

SUB.PROG. NAME	COMMENT
1	2

PROGRAM STATEMENTS

1

C3600 Data Entry System	TABLE CODING SHEET (SINGLE)	NOTES:	PAGE OF
	INITIALS: / DATE:		TABLE:

FIGURE 2.3.1

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

1 ARGUMENTS

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.

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

C3600 Data Entry System	TABLE CODING SHEET (DOUBLE)		NOTES:	PAGE OF
	INITIALS:	DATE:		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	
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.

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 (Δ).
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 (Δ).
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

C3600 Data Entry System

INITIALS:

DATE:

NOTES:

PAGE OF

FORMAT:

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	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				0										DEFINE XO1 4, DEFINE XO2 2,
KDATE				4	4	N	10							COMPUTE XO1 = KDATE,
														PERFORM DCHEK,
														IF XO1 = 0 THEN ALARM 'DATE OF PURCHASE WRONG',
BDATE				4	4	N	11							COMPUTE XO1 = BDATE,
														PERFORM DCHEK,
														IF XO1 = 0 THEN ALARM 'DATE OF PAYMENT WRONG',
														IF KDATE > BDATE THEN ALARM 'DATE OF PURCHASE GREATER THAN DATE OF PAYMENT',
AMOUN				10	1	N	12		O					
DISCO				9	9	N	13		O			N		IF KDATE = BDATE
														THEN COMPUTE DISCOUNT = AMOUNT/100*3
														ELSE COMPUTE DISCOUNT = 0,

, SS, AN, A - 10, FILL = , ∧, *, 0 - 13, KIND = ∧, N, C, D, I, K

3600 Data Entry System

SUBPROGRAM CODING SHEET

INITIALS:	DATE:	NOTES:	PAGE OF
			SUBPROGRAM:

EXAMPLE 3.1.4b

SUB.PROG. NAME	COMMENT
1	2
DCHEK	START: XO1 = YYMM, STOP: XO1 = WHEN WRONG DATE, XO2 IS WORKING REGISTER

PROGRAM STATEMENTS

1

```
        COMPUTE XO2 = XO1 MOD 100, NOTE XO2 = MM,
        COMPUTE XO1 = XO1 / 100 , NOTE XO1 = YY,
        IF (XO1 < 70) OR (XO1 > 80) THEN GOTO NOTOK,
        IF (XO2 < 1) OR (XO2 > 12) THEN GOTO NOTOK,
        NOTE THE DATE IS CORRECT,
        COMPUTE XO1 = 1,
        GOTO STOP,

NOTOK:
        COMPUTE XO1 = 0,

STOP:
        END,
```

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

Character	Meaning
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	GOTO	SET
ALLOW	IF	SKIP
AND	IN	SUBFORMAT
AT	INVALID	THEN
COMPUTE	LIMIT	TO
CONNECT	MOD	VALID
DEFINE	MOVE	X00
DISALLOW	NOT	X01
DISC	NOTE	X02
DISPLAY	OR	-
ELSE	PERFORM	-
END	SEARCH	-
FIELD	SELECT	X99
GIVING		

Table 3.2.1-2. List of Reserved Verbs

Verb	may be used in: Field Program	Subprogram
ALARM	x	x
ALLOW	x	x
COMPUTE	x	x
CONNECT	x	x
DEFINE	x	x
DISALLOW	x	x
DISPLAY	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

The arithmetic operators are used to perform specific arithmetic operations. The used symbols and their operation are:

+	Addition
-	Subtraction
*	Multiplication
/	Division
MOD	Modulo

3.2.3 Relational Operators

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

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

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		

3.3 Operands 3.3

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 number of significant digits in the data is greater than the register length, a runtime error will occur.

3.3.3 Fields

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

Field Definition Type	Operand Type
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	Justification	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	Justification	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

3.3.4 Subscripts

Subscripts are used to refer to individual characters in a register or a field. A subscript is a numeric constant. Subscripts should be in the range 1 to the length of the register or the field, where 1 is the left-most subscript.

Examples of subscripting are:

```
COMPUTE MONTH = DATE (3) * 10 + DATE (4),
COMPUTE X01(5) = 9,
MOVE 'A' TO X02(1),
```

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 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.

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 numeric 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 an arithmetic expression results in a numeric value. In the example the result of the evaluation is the sum of the values of register X01 and register X02.

The arithmetic operators allowed are:

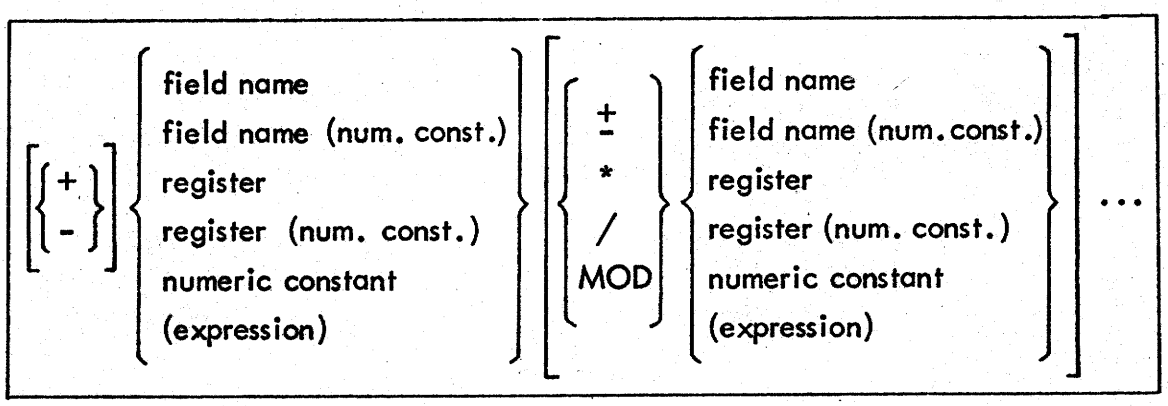
Operator	Meaning	
+	addition	called adding operators
-	subtraction	
*	multiplication	called multiplying operators
/	division	
MOD	modulo	

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:



Examples of arithmetic expressions are:

```
X01
FLD1
59
+X02(1)
-59
X01*FLD
X01(1)+X01(2)+X01(3)
(X01+X02)/2
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 : Multiplying operators (*, /, and MOD)
3 (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.

3.6 Conditions

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.

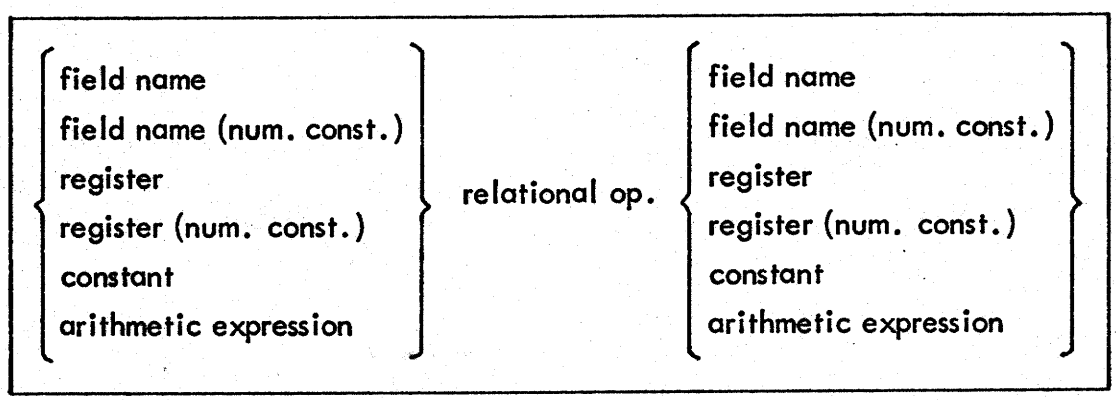
3.6.1 Relation

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:

Relational operator	Meaning
>	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.

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
```

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!

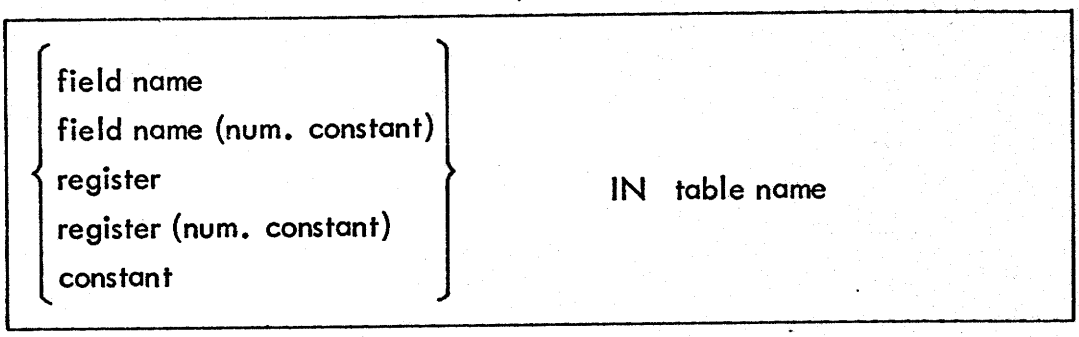
When nonnumeric operands are of unequal length (i.e., they do not contain the same number of character positions), the longest operand is treated as if the right-most characters were truncated, to make it the same length as the other operand. The comparison is then made as though they were the same length.

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.
'Δ39'	'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.

3.6.2 Table Condition

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.

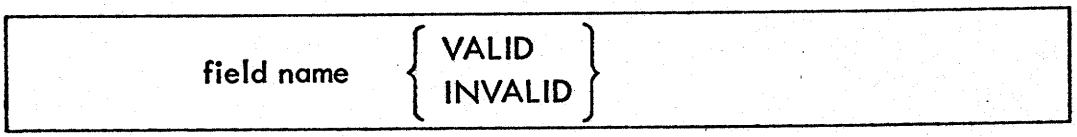
The operand preceding IN and the arguments in the table must be of the same type (i.e., either numeric or nonnumeric). The methods for comparing operands as described in Sections 3.6.1.1 and 3.6.1.2 are also used when evaluating a table condition.

Examples of table conditions:

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

3.6.3 Validity Condition

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

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

The logical operators and their meanings are:

Logical Operator	Meaning
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.

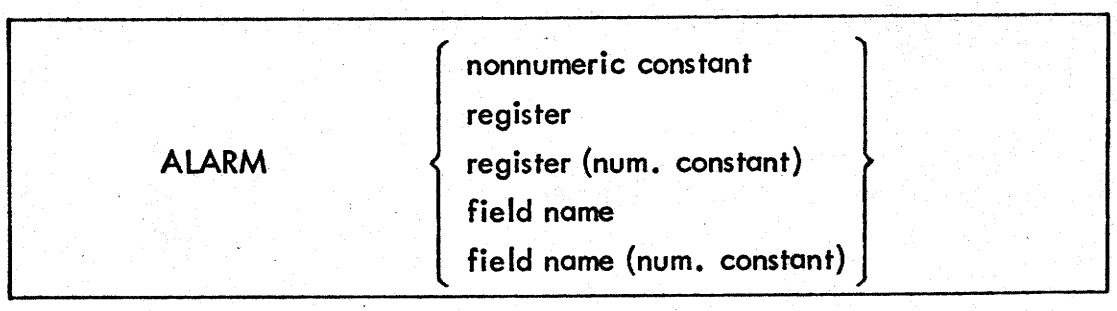
A field program consists of none, one or more statements.

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:



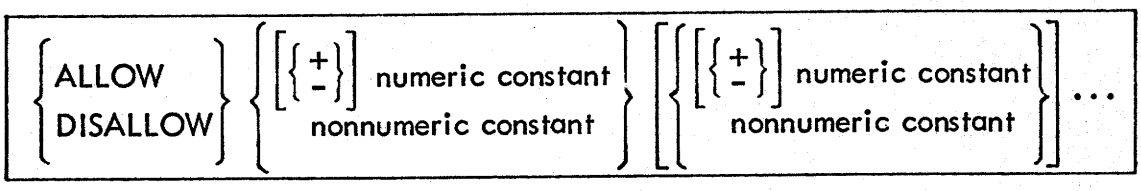
The ALARM statement displays the contents of the operand on the second line (the message part) of the keystation screen. Statements following the ALARM statement are not executed, and the operator must either correct or bypass the field.

Examples of the ALARM statement:

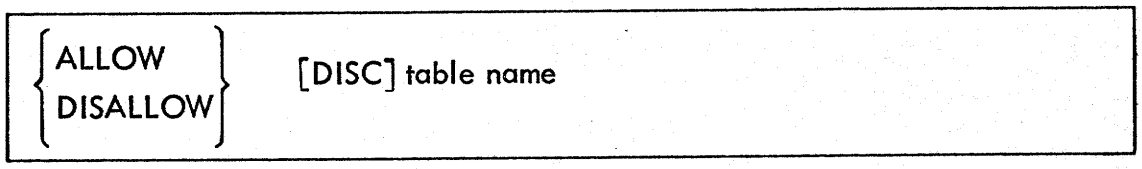
```
ALARM 'FINAL PRICE NOT OK',
CONNECT 'BATCH OUT OF BALANCE, DIFF=' TO X01 GIVING
X02,
ALARM X02,
```

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:



or:



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 double entried. 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

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

```
COMPUTE { current field name                    } = arithmetic expression
        { current field name (num. constant)    }
        { register                              }
        { register (num. constant)              }
```

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.

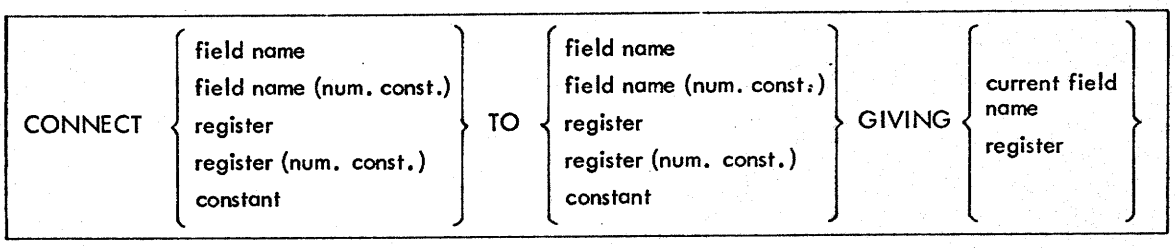
Examples of the COMPUTE statement:

```
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,
```

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

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,
```

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:

```
                 { nonnumeric constant
                 { register
DISPLAY          { register (num. constant)
                 { field name
                 { field name (num. constant)
```

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 END Statement

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
```

3.7.1.8 END SUBFORMAT Statement

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
```

3.7.1.9 GOTO Statement

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.

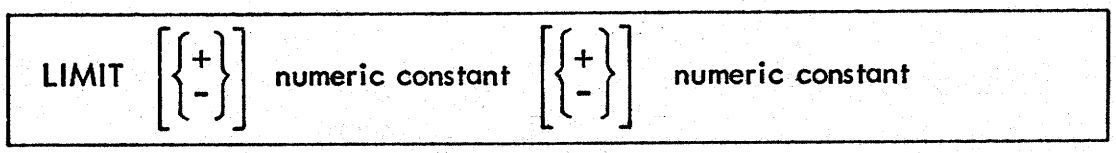
Examples of labels and the GOTO statement:

```
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:
```

3.7.1.10 LIMIT Statement 3.7.1.10

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,
```

3.7.1.11 MOVE Statement

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:

```
MOVE { field name                    } TO { current field name
     { field name (num. const.)      }    { current field name (num.const.)
     { nonnumeric constant           }    { register
     { register                      }    { register (num. const.)
     { register (num. const.)        }
```

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),
```

3.7.1.12 NOTE Statement

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.

3.7.1.13 PERFORM Statement 3.7.1.13

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,
```

3.7.1.14 SEARCH Statement

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

```
SEARCH { field name                    }
       { field name (num. const.)      }
       { register                      }  IN [DISC] table name
       { register (num. const.)        }
       { constant                      }

       GIVING { current field name                }
              { current field name (num. const.)  }
              { register                          }
              { register (num. const.)            }

       AT END  unconditional statement
```

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 be double-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 of the argument which matched is stored in the operand following the word GIVING, and the unconditional statement following AT END is not executed.

If the search is unsuccessful, control is transferred to the unconditional statement following the words AT END, and the operand after the word GIVING is not 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,
```

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

3.7.1.15. SELECT Statement 3.7.1.15

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,
```

3.7.1.16 SET Statement 3.7.1.16

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,
```

3.7.1.17 SKIP Statement 3.7.1.17

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,
```

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)
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

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
C0001	CHECK-DIGIT CONTROL - XO1=CUSTOMER NUMBER

PROGRAM STATEMENTS
1
IF (XO1(1)*5
+XO1(2)*4
+XO1(3)*3
+XO1(4)*2
+XO1(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	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
CUSNO				5	ON		1							COMPUTE XO1=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.

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.
* Right/left justification and insertion of fill characters.
* A typecheck is performed.
* Execution of the field program.

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.

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.

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.
* The field is incremented by 1.
* Right/left justification and insertion of fill characters.
* 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

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

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

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

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, 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.

4.7 Replay 4.7

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 kay has been used.

4.8 Execution of Format Image 4.8

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 TRANS, with the exception of DISC tables which are treated separately.

TRANS 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

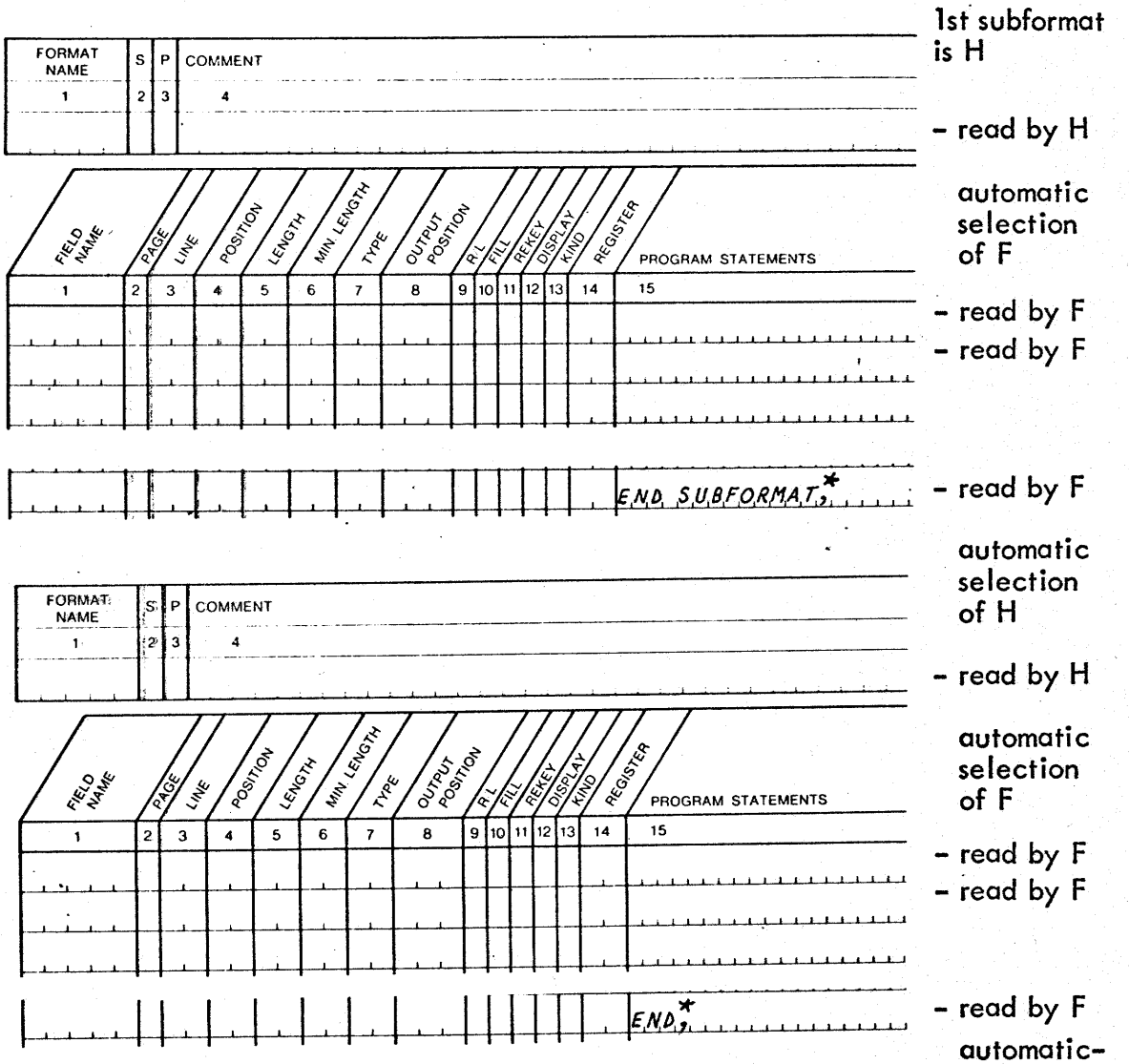
The format text of the coding sheets is keyed under the control of the standard format FORM.

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

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:



* 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 (not keyed field)	No check; automatic selection of subformat 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 either 'NΔ', 'SN', or 'SS', 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)	If LENGTH = 'ΔΔ', or LENGTH = 0: only PROGRAM STATEMENTS may be specified. The following applies if LENGTH > 0: - Either PAGE, LINE, and POSITION are all specified, or none of them. - MIN.LENGTH, TYPE, and OUTPUT POSITION are specified. - If REGISTER is not specified, then KIND is either 'N', 'K', or 'Δ'. - If REGISTER is specified, then KIND is either 'C', 'D', or 'I'. When PROGRAM STATEMENTS start with 'END SUBFORMAT,', subformat H is automatically selected. When PROGRAM STATEMENTS start with 'END,', subformat E is automatically selected.

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 FIELDNAME	F.1
ILLEGAL FORMATNAME	H.1
ILLEGAL SUBFORMAT NAME	H.2
KIND "I" ONLY ALLOWED IF TYPE = "N" - "SN" OR "SS"	F.13
MIN.LENGTH GREATER THAN FIELDLENGTH	F.6

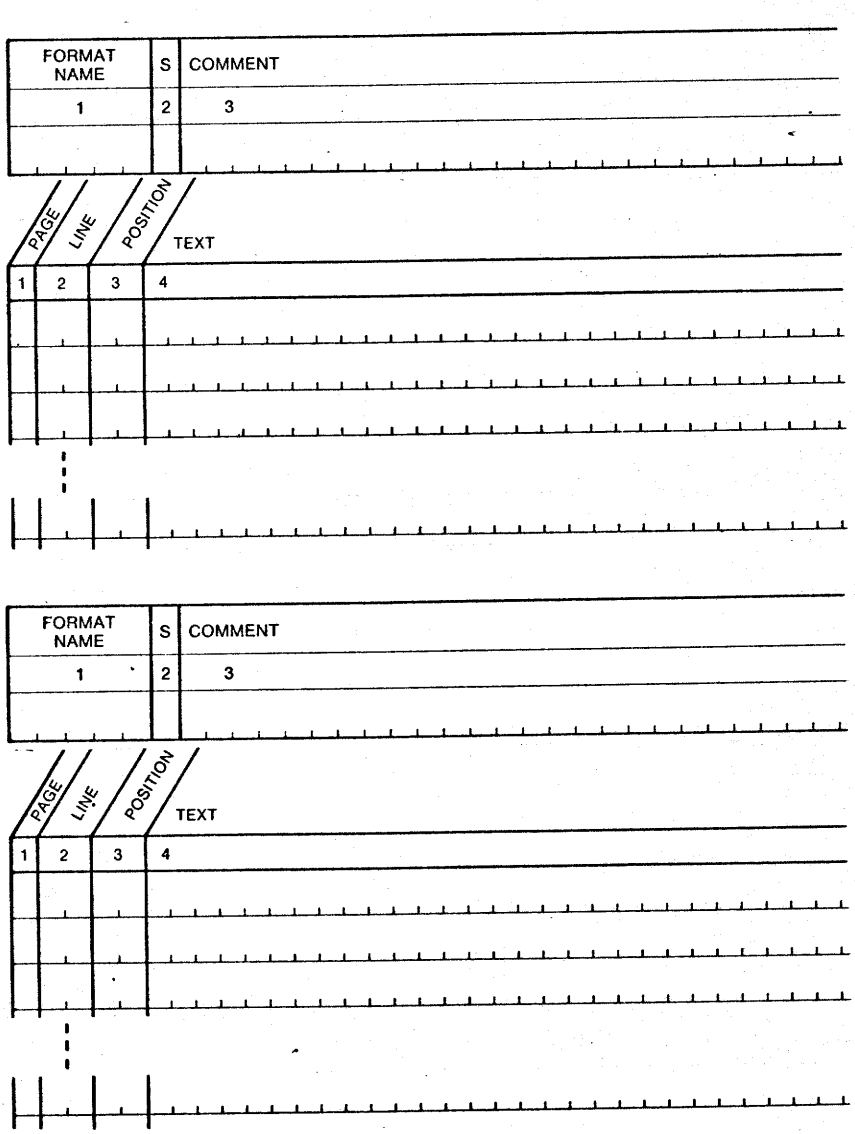
The standard format IMAGE contains 3 subformats (see Appendix IV).

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:



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 (not keyed field)	No check; automatic selection of sub-format 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 FORMATNAME	H.1
ILLEGAL SUBFORMATNAME	H.2

When the created batches are considered to be correct, translation of the format is initiated by activating the supervisor program TRANS (see User's Manual):

TRANS 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.

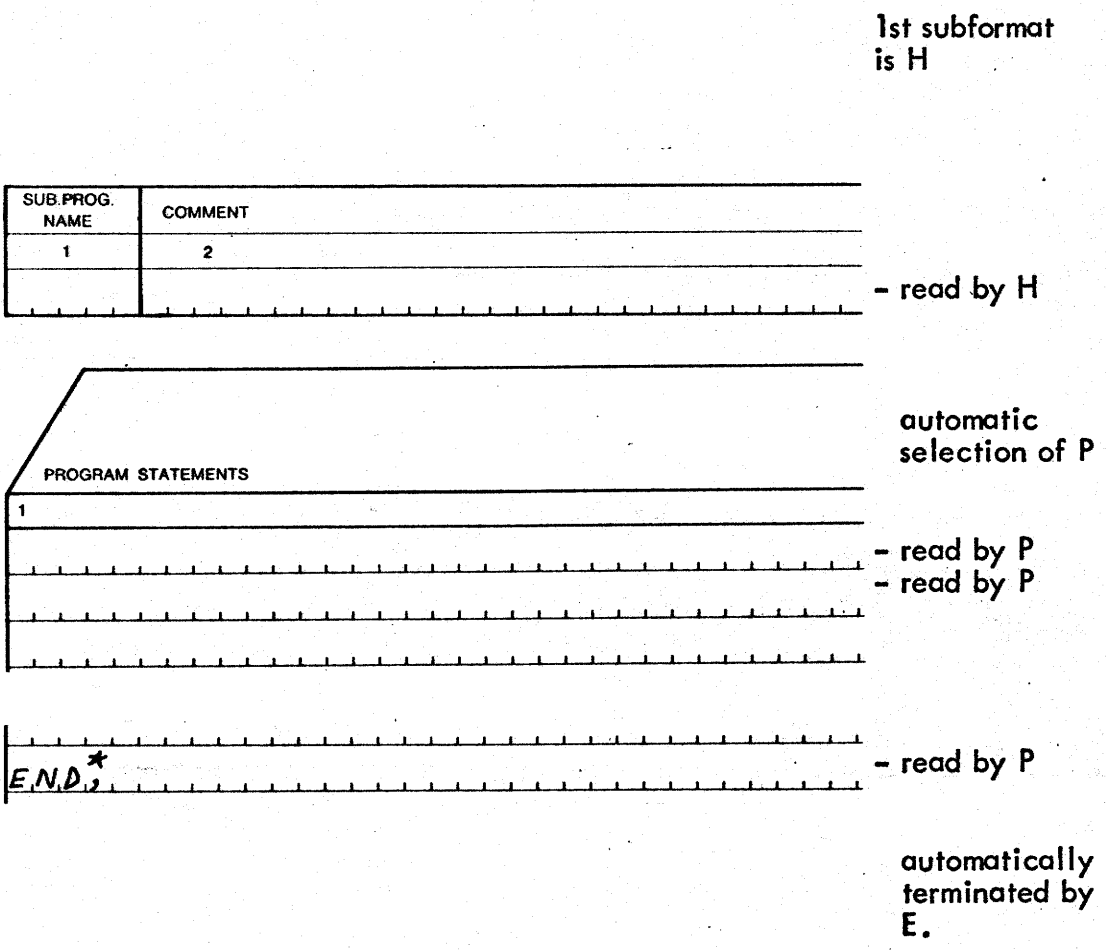
The SUBPR standard format contains 3 subformats (see Appendix IV):

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:



* Automatic selection of subformats is ackomplished 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 SUBPROGRAMNAME 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 SUBPROGRAMNAME	H.1

When the created batch is considered to be correct, translation of the subprogram is initiated by activating the supervisor program TRANS (see User's Manual):

```
TRANS SUBPR 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.

5.3.1 New Core Tables 5.3.1

The keying of the table text from the coding sheets is controlled by the standard format TABLE.

The TABLE standard format contains 8 subformats (see Appendix IV):

H - reads 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;
E - terminates batch (table text).

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

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

automatic selection of subformat 1 or 2

1 ARGUMENTS

- read by 1 or 2
- read by 1 or 2
- read by 1 or 2

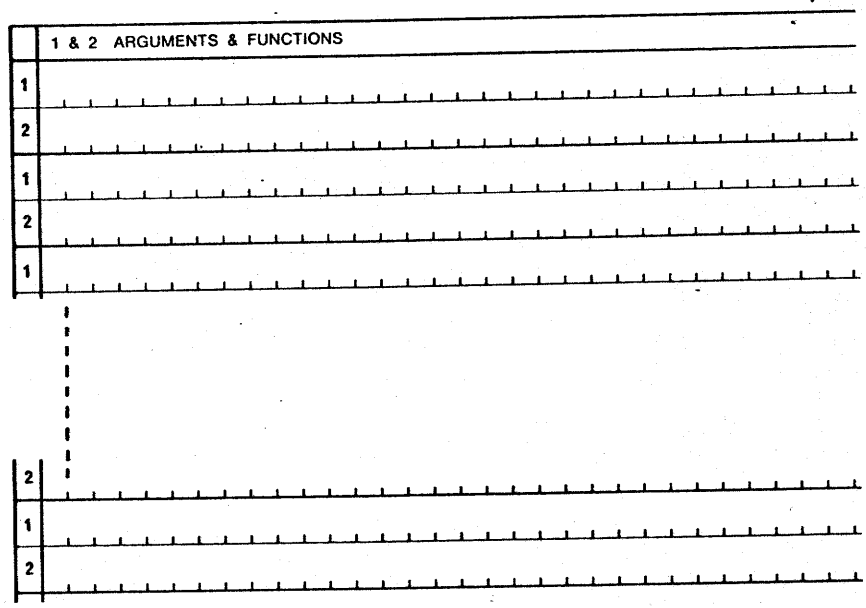
Manual selection of E terminates batch.

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 selection of subformat 3, 4, 5, or 6

- read by 3,4, 5, or 6

- read by 3,4, 5, or 6

- read by 3,4, 5, or 6

Manual selection of E terminates 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'.
2. TYPE (A)	'S' or 'D'
3. ARGUMENTTYPE (A) (A-TYPE)	'NΔ' or 'AN'
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) (F-TYPE)	'NΔ' or 'AN'
6. FUNCTIONLENGTH (N) (F-LGTH)	FUNCTIONLENGTH must be a number from 1 to 80.
7. Field description (not keyed field)	No check; depending on ARGUMENT-TYPE and FUNCTIONTYPE a subformat from 1 through 6 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 TABLENAME	H.1

When the created batch is considered to be correct, translation of the table is initiated by activating the supervisor program TRANS (see User's Manual):

```
TRANS TABLE   table-batch
```

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

5.3.2 Disc Tables

When using a table with many table elements, it is sometimes necessary to set up the table as a disc table.

The entering of such a disc table is ackomplished with the help of the supervisor program DISCT. This program checks if the table text is correctly structured, and, if so, translates the table into internal form, after which the table name is included in the disc-table library. This procedure allows the table to be referenced as an ordinary table.

Translating means that a hash-organized register, based on the table text, is generated on the disc. This generating (translating) 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.

At a later point in time it will be possible to insert, or to delete, elements in the translated table, so as to avoid repeating the translation procedure.

The table text is either keyed to a batch under the control of the standard format TABLE (see Section 5.3.1), or it may be stored on a magnetic tape generated by another computer.

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

The first block should contain information on the disc table that is to be created: (All information should be given in character form - 8 bit ASCII.)

Byte	Contents
1-6	Name of disc table. Only the first 5 characters are used for table identification. Fill characters: space.
7-9	Argument length + function length. 3 digits. If defined length is less than 3 digits, the preceding empty spaces are zero-filled.
10-11	Argument length. 2 digits, right-justified, zero-fill.
12-16	Number of table elements. 5 digits, right-justified, zero-fill.

Byte	Contents
17	Table type = single 'S', or double 'D'.
18-19	Argument type: 'NΔ' or 'AN'.
20-21	Function type: 'NΔ' or 'AN; defined only if table type = 'D'.
22-26	The word 'RCTAB'.
27-512	Not used.

Each of the following blocks must contain a complete number of table elements, as many as possible in each block.

When the table text is considered correct, the table is translated by activating the supervisor program DISCT through one of the two calls below: (See User's Manual.)

DISCT MT0	(if the table text is stored on magnetic tape)
DISCT table-batch	(if the table text is keyed under the control of the TABLE format)

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

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.

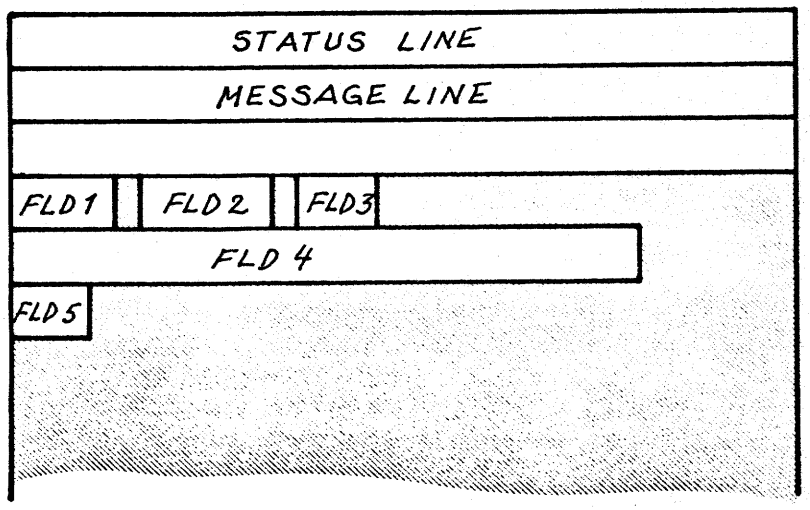
- If there are no more lines available, the next field is keyed from the left most position on the first data line of the screen, which then makes this field the first field on the subformat's next page. Previous to keying this field the screen's input section is blanked.

Example 6.1.1

A subformat starts with the following field descriptions:

FIELD NAME	PAGE	LINE	POSITION	LENGTH	MIN. LENGTH	TYPE	OUTPUT POSITION	R/L	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.

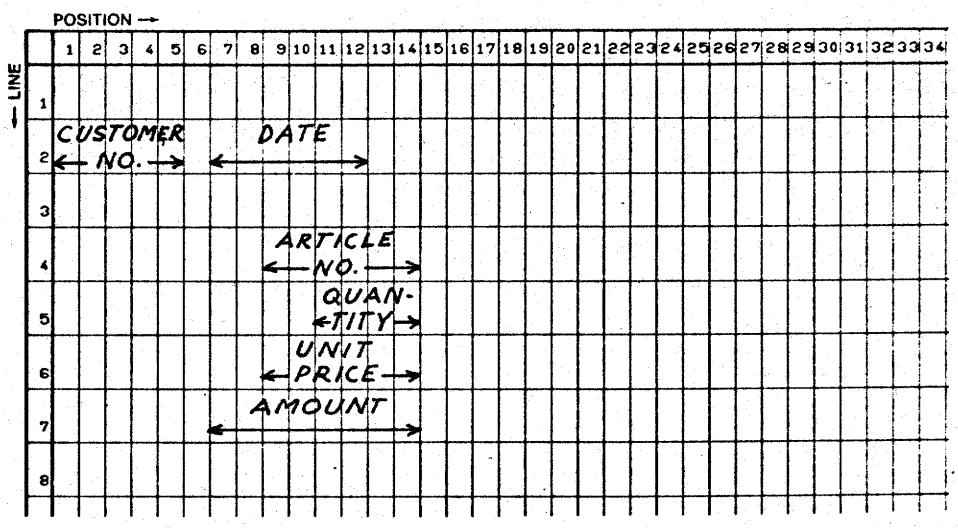
- If PAGE has the same value in the next field description as in the preceding one, the next field is keyed from the screen position specified under LINE and POSITION.

- 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	R/L	FILL	REKEY	DISPLAY	KIND	REGISTER	PROGRAM STAT
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CUSTN	1	2	1	5										
DATE	1	2	7	6										
ARTIN	1	4	9	6										
QTY	1	5	1	4										
UPRIC	1	6	9	6										
AMOUN	1	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
CUSTN	1	2	1	5										
DATE				6										
ARTIN	1	4	9	6										
QTY	1	5	11	4										
UPRIC	1	6	9	6										
AMOUN	1	7	7	8										

6.1.3 Defining Tags

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:

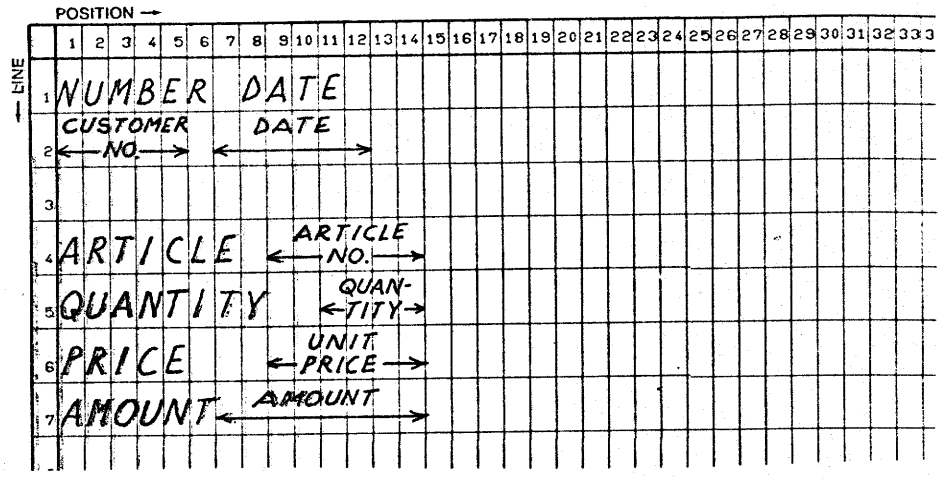
- On selection of a new subformat or a new page within a subformat the screen is blanked and all tags belonging to the new page in the subformat are laid out in their specified screen positions before keying is started to the first field of the page.

- 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:



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
EXAMP	1	N	THE SUBFORMAT IS USED AS A GUIDE

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	16	1	1	0	AM	0							
														SELECT SUBFORMAT 2,
														END SUBFORMAT

Matching tags:

FORMAT NAME	S	COMMENT
1	2	3
EXAMP	1	GUIDE

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

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 TRANS (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 User's Manual 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: ☐ customer no.: ☐ article no.: ☐ quantity: ☐

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	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							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	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
D.T.Y.P.E				2	2	A	1					N		MOVE 'AA' TO D.T.Y.P.E,
														NOTE TYPE OF DOCUMENT,
T.O.T.A.L				1.0	1.0	N	1.2		0			N		COMPUTE TOTAL =
														TOTAL + AMOUNT
D.A.T.E.				6	6	N	1.7					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 keyed 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

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	1,2					C	02	
:														
:														
CODE				4	1	AN	1,7					C	03	

The value of the DATE field in Subformat 2 will be the value registrated 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 ackomplished 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
1	2	3	4
EXAMP	2		

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
CUSTN				4	4	N	1					D	01	
ARTIN				3	2	N	2					D	02	
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.

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
EXAMP	2		

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
TYPE				1	1	N	1					I	01	
INF1				4	1	N	2							
INF2				4	1	N	3							END SUBFORMAT;

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:

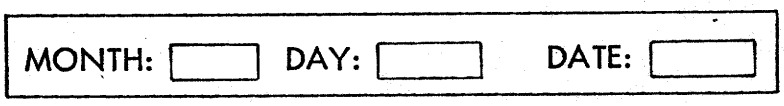
1. If one only wants to find out if a certain value exists, a single-entried table is used, listing all possible values.

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 double-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. See Chapter 5 for further details!

Example 6.6

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



Corresponding field descriptions:

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
MONTH				9	1	AN	10	L						IF NOT (MONTH IN MTABL)
														THEN ALARM 'MONTH IS WRONG',
DAY				9	1	AN	11	L						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	S	AN	9		

1 ARGUMENTS
MONDAY
TUESDAY
WEDNESDAY
THURSDAY
FRIDAY
SATURDAY
SUNDAY

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

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.

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

Program part

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
SKIP - FIELDS	3
IF - THEN	4-5
IF - THEN - ELSE	8-10
SET	1

Study the example on the opposite page!

FORMAT NAME	S	P	COMMENT
1	2	3	4
EXPL	1		REQUIRED SPACE - 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
F1				1	0	N	1							DEFINE X01 1,, COMPUTE X01=F1,,
														PERFORM CHECK,,
				0										SELECT SUBFORMAT 2,,
														END SUBFORMAT,,

1	2	3	4
EXPL	2		REQUIRED SPACE - 2

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

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 User's Manual 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 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 characters = 97 segments.

APPENDIX III

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.

1. Customer number	Discout percentage	Invoice number
2. Customer name		
3. Customer address		
4. Postal code	5. City	Date

1. Article no.	2. Article name	3. Quantity	4. Unit price	Discount	5. Final price

1. Total price

RC3600 Data Entry System — SCREEN LAYOUT

INITIALS: AA	DATE: 76.09.03	NOTES: HEAD OF INVOICE		PAGE 1 OF 3
		FORMAT: INVOI	SUBFORMAT: 1	PAGE: 1

POSITION →

```
LINE
 1  CUSTOMER NUMBER    CUSTOMER NAME
 2  XXXXXXXXXXXX       XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 3
 4  CUSTOMER ADDRESS
 5  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
 6
 7  POSTAL CODE        CITY
 8  XXXX               XXXXXXXXXXXXXXXXXX
 9
10
11
12
13
14
15
16
17
18
19
20
21
```

C3600 Data Entry System

SCREEN LAYOUT

INITIALS:	DATE:	NOTES:	PAGE
AA	76.09.03	ONE LINE IN BODY OF INVOICE	PAGE 2 OF 3

FORMAT:	SUBFORMAT:	PAGE:
INVOI	2	1

POSITION →

← LINE

```
1  ART NUMBER   ARTICLE NAME                QUANTITY  UNIT PRICE   DISCOUNT     FINAL PRICE
2  XXXXXXXXX    XXXXXXXXXXXXXXXXXXXXXXXX   XXXXX     XXXXXXXXXX   XXXXXXXXXX   XXXXXXXXX
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
```

C3600 Data Entry System — SCREEN LAYOUT

NOTES:	PAGE 3 OF 3
END OF INVOICE	

INITIALS:	DATE:	FORMAT:	SUBFORMAT:	PAGE:
AA	76.09.03	INVOI	3	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)

← LINE

1 TOTAL PRICE
2 XXXXXXXXXXXXXX

20 KEY YES IF YOU WANT TO START ON A NEW INVOICE
21 KEY NO IF YOU WANT TO FINISH KEYING

FORMAT CODING SHEET

C3600 Data Entry System

INITIALS: AA

DATE: 76.09.03

NOTES:

FORMAT: INVOI

FORMAT NAME	S	P	COMMENT
1	2	3	4
INVOI	H	Y	SUBFORMAT H - DEFINITION OF REGISTERS

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
				0										DEFINE XO1 3, NOTE DISCOUNT PERCENTAGE,
														DEFINE XO2 10, NOTE ARTICLE NUMBER,
														DEFINE XO3 10, NOTE FINAL PRICE,
														DEFINE XO4 15, NOTE TOTAL PRICE,
														DEFINE XO5 10, NOTE REGISTER FOR SEARCHING ARTICLE,
														SELECT SUBFORMAT 1,
														END SUBFORMAT,

7, TYPE = N, SN, [illegible] N, A - 10, FILL = ,^,*,0 - 13, KIND = ^,N, C, D, I, K

FORMAT CODING SHEET

C3600 Data Entry System

INITIALS: AA

DATE: 76.09.03

NOTES:

FORMAT: INVO1

FORMAT NAME	S	P	COMMENT
1	2	3	4
INVO1	1		SUBFORMAT 1 - HEAD OF INVOICE

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
CUSNO	1	2	1	12	8	AN	1	L						NOTE READ CUSTOMER NUMBER - SEARCH IN CUSTOMER TABLE FOR
														DISCOUNT PERCENTAGE AND DUMP I.T.,
														SEARCH CUSNO IN GIVING XO1 AT END
														ALARM 'CUSTOMER NUMBER NOT KNOWN',
DISCT				3	0	N	2					N		NOTE INSERT DISCOUNT PERCENTAGE IN RECORD,
														IF CUSNO VALID THEN COMPUTE DISCT= XO1 ELSE COMPUTE XO1= 0,
NAME	1	2	20	50	1	A	3	L						NOTE READ NAME,
ADDR	1	5	1	50	1	AN	4	L						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	6	L						NOTE READ CITY,

7, TYPE - N, SN, SS, AN, A - 10, FILL = , ,*,0 - 13, KIND = ^,N, C, D, I, K

C3600 Data Entry System

FORMAT CODING SHEET

INITIALS: AA

DATE: 76.09.03

NOTES:

FORMAT: INVOI

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	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
				0										NOTE, DUMMY FIELD FOR RESETTING REGISTER,
														COMPUTE X04= 0,
														SELECT SUBFORMAT 2,
														END SUBFORMAT,

7. TYPE = N, SN, [illegible] N, A - 10. FILL = , . , *, 0 13. KIND = △, N, C, D, I, K

FORMAT CODING SHEET

C3600 Data Entry System

INITIALS: AA | DATE: 76.09.03 | NOTES: | PAGE 4 OF 6 | FORMAT: INVOI

FORMAT NAME	S	P	COMMENT
1	2	3	4
INVOI	2		SUBFORMAT 2 - ONE LINE IN BODY OF INVOICE

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
ARTNO	1	2	1	10	10	N	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	AN	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	35	6	1	N	3							NOTE, READ QUANTITY,
UNIT	1	2	45	10	1	N	4							NOTE, READ UNIT PRICE,

7. TYPE = N, SN, SS, AN, A – 10. FILL = , ∧ , • , 0 – 13. KIND = ∧ , N, C, D, I, K

C3600 Data Entry System

FORMAT CODING SHEET

INITIALS: AA

DATE: 76.09.03

NOTES:

FORMAT: INV01

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	KIND	REGISTER	PROGRAM STATEMENTS
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DISCT	1	2	57	10	0		5				Y	N		NOTE, COMPUTE PRICE AND DUMP IT -
														COMPUTE DISCOUNT AND INSERT IN RECORD,
														COMPUTE X03= QTY * UNITP, COMPUTE DISCT= X01 * X03 / 100,
FINAL	1	2	69	10	1	N	6							NOTE, READ FINAL PRICE AND CHECK AGAINST COMPUTED PRICE,
														IF FINAL <> X03 - DISCT THEN
														ALARM 'FINAL PRICE NOT OK',
														NOTE, UPDATE TOTAL PRICE,
														COMPUTE X04= X04 + FINAL,
														END SUBFORMAT,

7, TYPE = N, SN, S[illegible] A - 10, FILL = ,^,*,0 - 13, KIND = ^,N, C, D, I, K

C3600 Data Entry System

FORMAT CODING SHEET

INITIALS: AA

DATE: 76.09.03

NOTES:

FORMAT: INVO1

FORMAT NAME	S	P	COMMENT
1	2	3	4
INVO1	3		SUBFORMAT 3 - END OF INVOICE

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
TOTAL	1	2	64	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	1	21	64	3	2	A	0	L						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,

7, TYPE = N, SN, SS, AN, A - 10, FILL = , , *, 0 - 13, KIND = , N, C, D, I, K

C3600 Data Entry System	IMAGE CODING SHEET		NOTES:	PAGE 1 OF 3
	INITIALS: AA	DATE: 76.09.03	IMAGE TO HEAD OF INVOICE	FORMAT: INVOI

FORMAT NAME	S	COMMENT
1	2	3
INVOI	1	IMAGE TO HEAD OF INVOICE

PAGE	LINE	POSITION	TEXT
1	2	3	4
1	1	1	CUSTOMER NUMBER
1	1	20	CUSTOMER NAME
1	4	1	CUSTOMER ADDRESS
1	7	1	POSTAL CODE
1	7	20	CITY

C3600 Data Entry System

IMAGE CODING SHEET

INITIALS: AA

DATE: 76.09.03

NOTES: IMAGE TO ONE LINE IN BODY

PAGE 2 OF 3

FORMAT: 2

FORMAT NAME	S	COMMENT
1	2	3
INVOI	2	IMAGE TO ONE LINE IN BODY OF INVOICE

PAGE	LINE	POSITION	TEXT
1	2	3	4
1	1	1	ART NUMBER ARTICLE NAME QUANTITY UNIT PRICE DISCOUNT FINAL PRICE
1	21	1	SELECT SUBFORMAT 3 WHEN THE BODY OF INVOICE IS FINISHED

C3600 Data Entry System	IMAGE CODING SHEET		NOTES:	PAGE 3 OF 3
	INITIALS: AA	DATE: 76.09.03	IMAGE TO END OF INVOICE	FORMAT: INVOI

FORMAT NAME	S	COMMENT
1	2	3
INVOI	3	IMAGE TO END OF INVOICE

PAGE	LINE	POSITION	TEXT
1	2	3	4
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

C3600 Data Entry System	SUBPROGRAM CODING SHEET		NOTES:	PAGE 1 OF 1
	INITIALS: AA	DATE: 76.09.03	SUBPROGRAM USED IN INVOICE	SUBPROGRAM: CHE10

SUB.PROG. NAME	COMMENT
1	2
CHE10	SUBPROGRAM - CHECK DIGIT VERIFICATION (MODULUS 10) - XO2 = ARTICLE NUMBER

PROGRAM STATEMENTS

1

```
COMPUTE XO2= (XO2(1)*2
           +XO2(2)
           +XO2(3)*2
           +XO2(4)
           +XO2(5)*2
           +XO2(6)
           +XO2(7)*2
           +XO2(8)
           +XO2(9)*2
           +XO2(10)) MOD 10,
IF XO2 <> 0 THEN ALARM 'ERROR IN ARTICLE NUMBER',
END,
```

TABLE CODING SHEET (DOUBLE)

3600 Data Entry System

INITIALS: AA

DATE: 76.09.03

NOTES: ARTICLE TABLE USED IN INVOICE

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	10,10,10,10,10
1	PEAR
2	10,10,10,10,60
1	ONION
2	1,2,3,4,5,6,7,8,9,0
1	HORSE-RADISH
2	2,3,4,5,6,7,8,9,0,6
1	
2	
1	
2	
1	
2	
1	
2	
1	
2	

C3600 Data Entry System

TABLE CODING SHEET (DOUBLE)

INITIALS: AA

DATE: 76.09.0

NOTES: CUSTOMER TABLE USED IN INVOICE

TABLE: CTABL

TABLE NAME	T	A-TYPE	A-LGTH.	F-TYPE	F-LGTH.
1	2	3	4	5	6
CTABL	D	AN	12	N	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	

APPENDIX IV

Standard Formats FORM, IMAGE, SUBPR, and TABLE

```
LIST:  FORMAT-SOURCE TEXT  (B0C55):

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

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  H 001 001 0002 NAME           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 SUBFN          1  1 AN   2          NOTE  THE SUBFORMAT NAME IS 1 CHARACTER,
  H 002 002 0006                                       IF NOT(((SUBFN>='0') AND (SUBFN<='9')) OR
  H 002 003 0007                                              ((SUBFN>='A') AND (SUBFN<='Z'))) THEN
  H 002 004 0008                                       ALARM 'ILLEGAL SUBFORMAT NAME',
  H 003 001 0009 PROCT          1  0 A    3          NOTE SUBFORMAT PROTECTION AGAINST MANUAL SELECTION:
  H 003 002 0010                                       Y - NO MANUAL SELECTION  N - (DEFAULT) MANUAL SELECTION IS POSSIBLE,
  H 003 003 0011                                       ALLOW 'Y' 'N' ' ',
  H 004 001 0012 COMM          74  0 AN   4 L
  H 005 001 0013                0                      NOTE SUBFORMAT HEAD IS FINISHED
  H 005 002 0014                                        - CHANGE SUBFORMAT TO FIELD DESCRIPTION,
  H 005 003 0015                                       DEFINE X01 16, COMPUTE X01=0, NOTE BLANK MASK,
  H 005 004 0016                                       DEFINE X02 5, MOVE '     ' TO X02, NOTE CONSTANT - SPACES,
  H 005 005 0017                                       DEFINE X03 1, COMPUTE X03 = 0,
  H 005 006 0018                                       NOTE KEEPS TRACK OF PAGENO - LINENO AND POSITION,
  H 005 007 0019                                       DEFINE X04 1, COMPUTE X04 = 1, NOTE LAST USED PAGE,
  H 005 008 0020                                       DEFINE X05 1, NOTE  USED WHEN PAGE IS CHECKED,
  H 005 009 0021                                       SELECT SUBFORMAT F,
  H 005 010 0022                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0023 FORM  F N FIELD DESCRIPTION ENTRY

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  F 001 001 0024 FLDNM          5  0 AN   1 L          NOTE  FIELD NAME IS 0 TO 5 CHARACTERS,
  F 001 002 0025                                       IF ((FLDNM(1)<'A') OR (FLDNM(1)>'Z')) AND (FLDNM<>X02) THEN
  F 001 003 0026                                       ALARM 'ILLEGAL FIELDNAME',
  F 002 001 0027 PAGE           1  0 N    2          NOTE PAGENO IN DISPLAY-LAYOUT - FROM 1 TO 8,
  F 002 002 0028                                       CONNECT PAGE TO '' GIVING X05,
  F 002 003 0029                                       IF X05 = ' ' THEN SKIP 2 FIELDS; GOTO PEND,
  F 002 004 0030                                       LIMIT 1 8, COMPUTE X01(2)=1,
  F 002 005 0031                                       IF PAGE<X04 THEN  ALARM 'CURRENT PAGENO LESS THAN PREVIOUS PAGENO'
  F 002 006 0032                                       ELSE COMPUTE X04 = PAGE,
  F 002 007 0033                                       PEND:  NOTE END PAGENO - PROGRAM,
  F 003 001 0034 LINE           2  0 N    3 R        NOTE LINENO IN DISPLAY-LAYOUT - FROM 1 TO NO OF LINES ON DISPLAY,
  F 003 002 0035                                       LIMIT 1 21, COMPUTE X01(3)=1,
  F 004 001 0036 POS            2  0 N    4 R        NOTE POSITION IN LINENO DISPLAY-LAYOUT -
  F 004 002 0037                                       FROM 1 TO NO OF POSITIONS PER LINE ON DISPLAY,
  F 004 003 0038                                       LIMIT 1 80, COMPUTE X01(4)=1,
  F 005 001 0039 LENGT          2  0 N    5 R        NOTE  THE FIELD LENGTH IS FROM 0 TO 80,
  F 005 002 0040                                       IF LENGTH = 0 THEN SKIP 9 FIELDS ELSE LIMIT 0 80,
  F 005 003 0041                                       COMPUTE X01(5)=1,
  F 006 001 0042 MINLE          2  0 N    6 R        NOTE  THE MINIMUM LENGTH OF THE KEYED DATA,
  F 006 002 0043                                       IF MINLE>LENGTH THEN
  F 006 003 0044                                       ALARM 'MINLENGTH GREATER THAN FIELD LENGTH'
  F 006 004 0045                                       ELSE
  F 006 005 0046                                       LIMIT 0 80,
  F 006 006 0047                                       COMPUTE X01(6)=1,
  F 007 001 0048 TYPE           2  0 A    7 L        NOTE  THE FIELD TYPE IS NUMERIC - SIGNED NUMERIC
  F 007 002 0049                                             SPECIAL SIGNED NUMERIC - ALPHANUMERIC OR ALPHABETIC,
  F 007 003 0050                                       ALLOW 'N ' 'SN' 'S$' 'AN' 'A ',
  F 008 001 0051 OUTPS          3  0 N    8 R        NOTE  THE FIELD NUMBER OF THE FIELD IN OUTPUT BUFFER,
  F 008 002 0052                                       LIMIT 0 255, COMPUTE X01(8)=1,
  F 009 001 0053 JUST           1  0 A    9          NOTE  RIGHT OR LEFT JUSTIFICATION IN THE FIELD,
  F 009 002 0054                                       ALLOW 'L' 'R' ' ',
  F 010 001 0055 FILL           1  0 AN  10          NOTE  FILL CHARACTER FOR NOT USED POSITIONS IN THE FIELD,
  F 010 002 0056                                       ALLOW ' ' '0' '*',
  F 011 001 0057 VERIF          1  0 A   11          NOTE  VERIFICATION OF THE FIELD IN REKEY-MODE,
  F 011 002 0058                                       ALLOW 'N' 'Y' ' ',
  F 012 001 0059 DISP           1  0 A   12          NOTE  DISPLAY OF THE KEYED FIELD,
  F 012 002 0060                                       ALLOW 'N' 'Y' ' ',
```

```
F 013 001 0061 KIND            1  0 A   13
F 013 002 0062
F 013 003 0063
F 013 004 0064
F 013 005 0065
F 013 006 0066
F 013 007 0067
F 014 001 0068 REGIS           2  0 N   14 R
F 014 002 0069
F 015 001 0070 PROG           80  0 AN  15 L
F 016 001 0071                 0
F 016 002 0072
F 016 003 0073
F 016 004 0074
F 016 005 0075
F 016 006 0076
F 016 007 0077
F 016 008 0078
F 016 009 0079
F 016 010 0080
F 016 011 0081
F 016 012 0082
F 016 013 0083
F 016 014 0084
F 016 015 0085
F 016 016 0086
F 016 017 0087
F 016 018 0088
F 016 019 0089
F 016 020 0090
F 016 021 0091
F 016 022 0092
F 016 023 0093
F 016 024 0094
F 016 025 0095
F 016 026 0096
F 016 027 0097
F 016 028 0098
F 016 029 0099
F 016 030 0100
F 016 031 0101
F 016 032 0102
F 016 033 0103
F 016 034 0104
F 016 035 0105
F 016 036 0106
F 016 037 0107
F 016 038 0108
F 016 039 0109
F 016 040 0110
F 016 041 0111
F 016 042 0112
F 016 043 0113
F 016 044 0114
F 016 045 0115
F 016 046 0116
```

```
NOTE THE KIND IS DUPLICATION - INCREMENT
     - CONSTANT - NOT KEYED OR KEYED,
ALLOW 'N' 'C' 'D' 'I' 'K' ' ',
IF ((KIND='I') AND ((TYPE <> 'N') AND (TYPE <> 'SN')
  AND (TYPE <> 'SS'))) THEN
  ALARM 'KIND "I" ONLY ALLOWED IF TYPE = "N" - "SN" OR "SS"',
NOTE  REGISTER TO HOLD DUPLICATION OR CONSTANT VALUE,
LIMIT 1 99, COMPUTE X01(14)=1,
NOTE  PROGRAMSTATEMENTS,
NOTE  CHECK CHECKBOX CONTENTS,
COMPUTE X03 = X01(2) + X01(3) +X01(4),
IF X01(5)=0 THEN  GOTO EMPTY; NOTE  LENGTH NOT KEYED,
IF LENGTH=0 THEN  GOTO CONTINUE; NOTE  LENGTH = 0,
NOTE 'NORMAL'-CHECKBOX,
IF ((X03<>0) AND (X03<>3)) THEN GOTO ERROR;
NOTE ERROR IN PAGENO - LINENO OR POSITION,
IF X01(6)=0 THEN GOTO ERROR;  NOTE  MINKEYET NOT KEYED,
IF TYPE=X02 THEN  GOTO ERROR; NOTE  TYPE HAS NO VALUE,
IF X01(8)=0 THEN  GOTO ERROR; NOTE  OUTPOS NOT KEYED,
IF X01(14)=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 = DUPLIKATION,
IF KIND='C' THEN GOTO ERROR;  NOTE KIND = CONSTANT,
GOTO OK,
NOTE  'EMPTY' / 'CONTINUATION' CHECKBOX,
EMPTY: CONTINUE:
IF FLDNM<>X02 THEN  GOTO ERROR; NOTE  FIELD NAME KEYED,
IF X03 <> 0 THEN GOTO ERROR;
NOTE ERROR IN PAGENO - LINENO OR POSITION,
IF X01(6)=1 THEN GOTO ERROR;  NOTE MINKEYED KEYED,
IF TYPE<>X02 THEN  GOTO ERROR; NOTE  TYPE KEYED,
IF X01(8)=1 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 VERIF<>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 X01(14)=1 THEN GOTO ERROR;  NOTE REGISTER KEYED,
GOTO OK,
NOTE  ERROR IN CHECKBOX CONTENTS,
ERROR:
ALARM 'ERROR IN CHECKBOX CONTENTS',
NOTE  CHECKBOX CONTENTS OK,
OK:
COMPUTE X01=0,
IF PROG='END SUBFORMAT,' THEN  SELECT SUBFORMAT H,
IF PROG='END,' THEN  SELECT SUBFORMAT E,
END SUBFORMAT,
```

```
          REC  FNAME $ P COMMENT
          0117 FORM  E   END FORMAT FORMAT

$ FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

E 001 001 0118               0                    END,

END LIST
```

```
LIST:  FORMAT-SOURCE TEXT  (80052):

          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 V D K RG PROGRAM STATEMENTS

  H 001 001 0002 NAME          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 COMM         74  0 AN   2 L
  H 003 001 0005               0                        SELECT SUBFORMAT P,
  H 003 002 0006                                        END SUBFORMAT,

          REC  FNAME S P COMMENT
          0007 SUBPR P N SUBPROGRAM FORMAT - PROGRAMPARTS

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  P 001 001 0008 PROG         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 V D K RG PROGRAM STATEMENTS

  E 001 001 0011               0                        END,

END LIST
```

```
LIST:  FORMAT-SOURCE TEXT  (B0054):

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

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  H 001 001 0002 NAME           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 SUBFN          1  1 AN   2            NOTE THE SUBFORMAT NAME IS 1 CHARACTER,
  H 002 002 0006                                       IF NOT (((SUBFN>='0') AND (SUBFN<='9')) OR
  H 002 003 0007                                              ((SUBFN>='A') AND (SUBFN<='Z'))) THEN
  H 002 004 0008                                       ALARM 'ILLEGAL SUBFORMAT NAME',
  H 003 001 0009 COMM          74  0 AN   3 L
  H 004 001 0010                0                      NOTE SUBFORMAT HEAD IS FINISHED -
  H 004 002 0011                                            CHANGE SUBFORMAT TO TEXT DESCRIPTION,
  H 004 003 0012                                       DEFINE X01 1, COMPUTE X01=1, NOTE LAST USED PAGE,
  H 004 004 0013                                       SELECT SUBFORMAT F,
  H 004 005 0014                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0015 IMAGE F N TEXT DESCRIPTION ENTRY

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  F 001 001 0016 PAGE           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                                       ELSE
  F 001 006 0021                                         COMPUTE X01 = PAGE,
  F 002 001 0022 LINE           2  1 N    2 R          NOTE LINENO IN DISPLAY-LAYOUT - FROM 1 TO NO OF LINES ON DISPLAY,
  F 002 002 0023                                       LIMIT 1 21,
  F 003 001 0024 POS            2  1 N    3 R          NOTE POSITION IN LINENO IN DISPLAY-LAYOUT - FROM 1 TO NO OF
  F 003 002 0025                                            POSITIONS PER LINE ON DISPLAY,
  F 003 003 0026                                       LIMIT 1 80,
  F 004 001 0027 TEXT          80  1 AN   4 L          NOTE THE DISPLAY TEXT IS 1 TO 80 CHARACTERS,
  F 004 002 0028                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0029 IMAGE E N END IMAGE

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  E 001 001 0030                0                      END,

END LIST
```

```
LIST:  FORMAT-SOURCE TEXT  (80053):

           REC  FNAME S P COMMENT
           0001 TABLE H N  TABLE FORMAT - HEAD

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  H 001 001 0002 NAME           5  1 AN   1 L           NOTE  TABLENAME IS 1 TO 5 CHARACTERS,
  H 001 002 0003                                       IF (NAME(1)<'A') OR (NAME(1)>'Z') THEN
  H 001 003 0004                                         ALARM 'ILLEGAL TABLE NAME',
  H 002 001 0005 TYPE           1  1 A    2            NOTE  TABLETYPE IS EITHER SINGLE OR DOUBLE,
  H 002 002 0006                                       ALLOW 'S' 'D',
  H 003 001 0007 ARGT           2  1 A    3 L          NOTE  ARGUMENTTYPE IS NUMERIC OR ALPHANUMERIC,
  H 003 002 0008                                       ALLOW 'N ' 'AN',
  H 004 001 0009 ARGL           2  1 N    4 R O        NOTE  ARGUMENTLENGHT IS FROM 1 TO 80,
  H 004 002 0010                                       LIMIT 1 80,
  H 004 003 0011                                       IF TYPE= 'S' THEN
  H 004 004 0012                                         SKIP 2 FIELDS,
  H 005 001 0013 FUNCT          2  1 A    5 L          NOTE  FUNCTION TYPE IS NUMERIC OR ALPHANUMERIC,
  H 005 002 0014                                       ALLOW 'N ' 'AN',
  H 006 001 0015 FUNCL          2  1 N    6 R O        NOTE FUNCTION LENGTH IS FROM 1 TO 80,
  H 006 002 0016                                       LIMIT 1 80,
  H 007 001 0017                0                      NOTE SELECT ACTUAL SUBFORMAT,
  H 007 002 0018                                       IF TYPE='D' THEN  GOTO LD,
  H 007 003 0019                                         IF ARGT='N ' THEN  SELECT SUBFORMAT 1
  H 007 004 0020                                                    ELSE SELECT SUBFORMAT 2,
  H 007 005 0021                                         GOTO LEND,
  H 007 006 0022                                       LD:
  H 007 007 0023                                         IF ARGT<>'N ' THEN GOTO LDAN,
  H 007 008 0024                                           IF FUNCT='N ' THEN SELECT SUBFORMAT 3
  H 007 009 0025                                                        ELSE  SELECT SUBFORMAT 4,
  H 007 010 0026                                           GOTO LEND,
  H 007 011 0027                                       LDAN:
  H 007 012 0028                                           IF FUNCT='N ' THEN SELECT SUBFORMAT 5
  H 007 013 0029                                                        ELSE  SELECT SUBFORMAT 6,
  H 007 014 0030                                       LEND:
  H 007 015 0031                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0032 TABLE 1 N  TABLE FORMAT - SINGLE ENTRY - ARGT=N

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  1 001 001 0033 ARG            80  1 N    1 R O        NOTE  READ ARGUMENT,
  1 001 002 0034                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0035 TABLE 2 N  TABLE FORMAT - SINGLE ENTRY - ARGT=AN

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  2 001 001 0036 ARG            80  1 AN   1 L          NOTE  READ ARGUMENT,
  2 001 002 0037                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0038 TABLE 3 N  TABLE FORMAT - DOUBLE ENTRY - ARGT=N - FUNCT=N

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  3 001 001 0039 ARG            80  1 N    1 R O        NOTE  READ ARGUMENT,
  3 002 001 0040 FUNC           80  1 N    2 R O        NOTE  READ FUNCTION,
  3 002 002 0041                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0042 TABLE 4 N  TABLE FORMAT - DOUBLE ENTRY - ARGT=N - FUNCT=AN

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  4 001 001 0043 ARG            80  1 N    1 R O        NOTE  READ ARGUMENT,
  4 002 001 0044 FUNC           80  1 AN   2 L          NOTE  READ FUNCTION,
  4 002 002 0045                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0046 TABLE 5 N  TABLE FORMAT - DOUBLE ENTRY - ARGT=AN - FUNCT=N

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  5 001 001 0047 ARG            80  1 AN   1 L          NOTE  READ ARGUMENT,
  5 002 001 0048 FUNC           80  1 N    2 R O        NOTE  READ FUNCTION,
  5 002 002 0049                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0050 TABLE 6 N  TABLE FORMAT - DOUBLE ENTRY - ARGT=AN - FUNCT=AN

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  6 001 001 0051 ARG            80  1 AN   1 L          NOTE  READ ARGUMENT,
  6 002 001 0052 FUNC           80  1 AN   2 L          NOTE  READ FUNCTION,
  6 002 002 0053                                       END SUBFORMAT,

           REC  FNAME S P COMMENT
           0054 TABLE E N  TABLE FORMAT - END

  S FLD LIN REC  NAME  P LN PS LG ML TY OUT J F V D K RG PROGRAM STATEMENTS

  E 001 001 0055                0                      END,

END LIST
```

APPENDIX V

Format Language Syntax

Operand: { fieldname | fieldname (num. constant) | register | register (num. constant) | constant }

Constant: { numeric constants | nonnumeric constants }

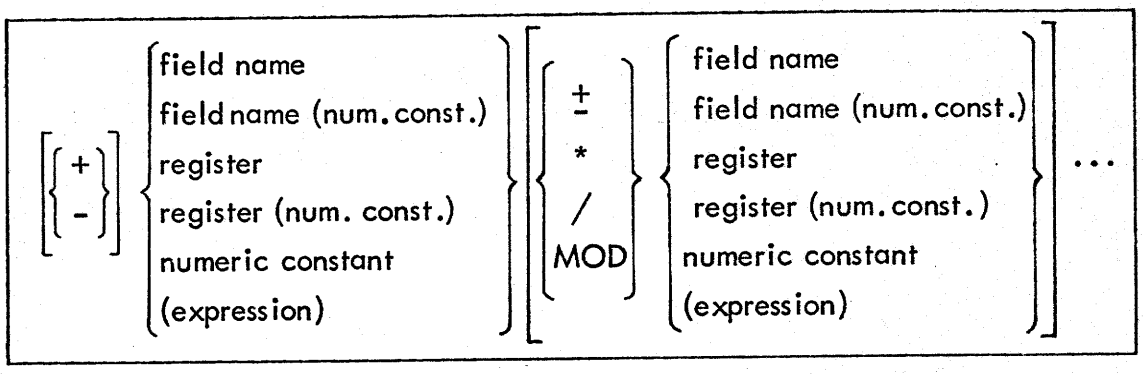
Operator: { arithmetic operator | relational operator | logical operator }

Arithmetic operator: { + | - | * | / | MOD }

Relational operator: { > | >= | = | < | <= | <> }

Logical operator: { AND | OR | NOT }

Expressions:



Conditions:

- relational condition
- table condition
- validity condition
- compound condition

Relational condition:

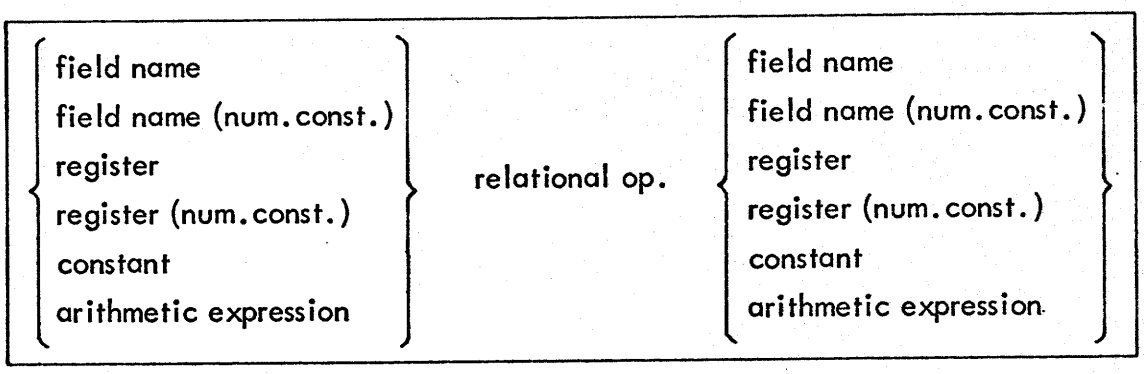
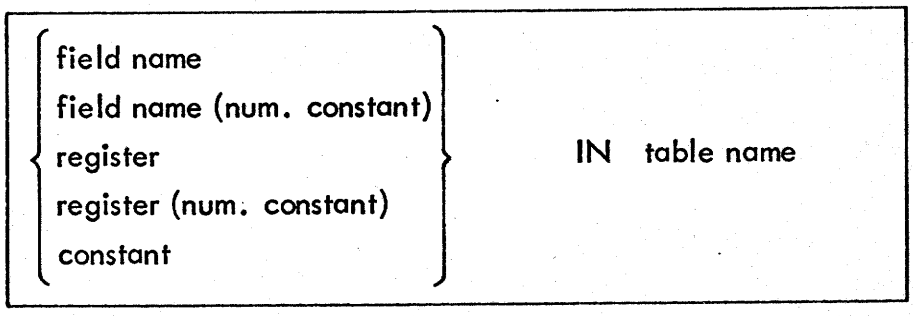
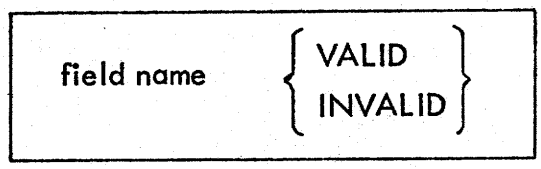


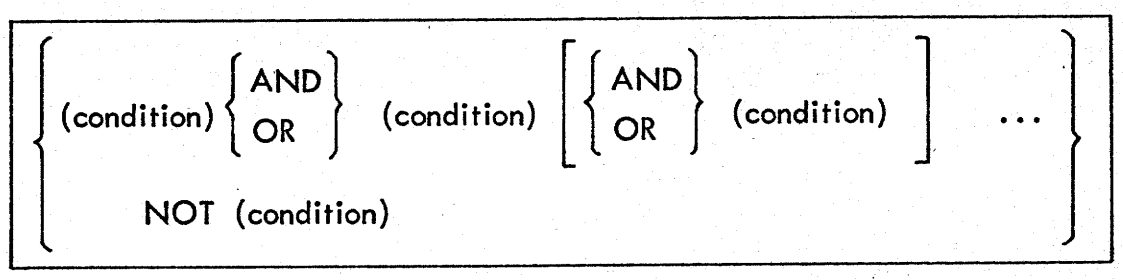
Table condition:



Validity condition:

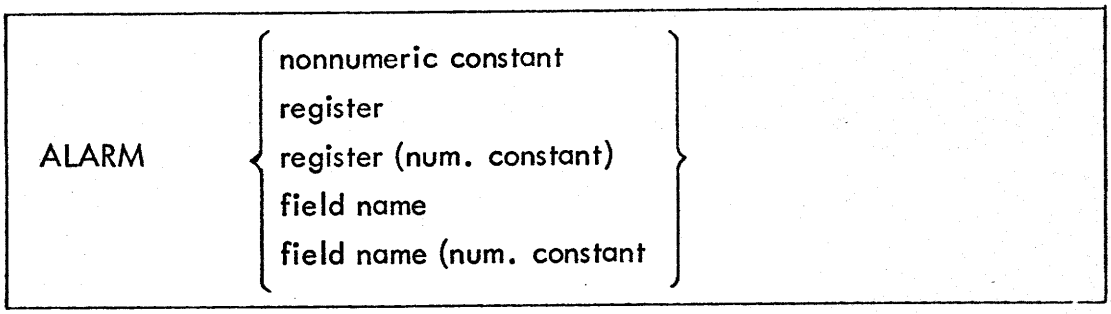


Compound condition:

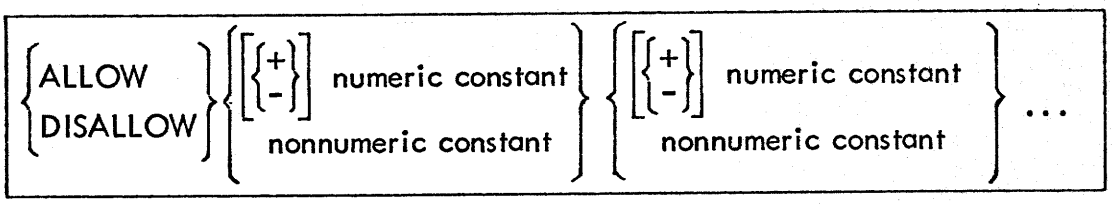


Statements:

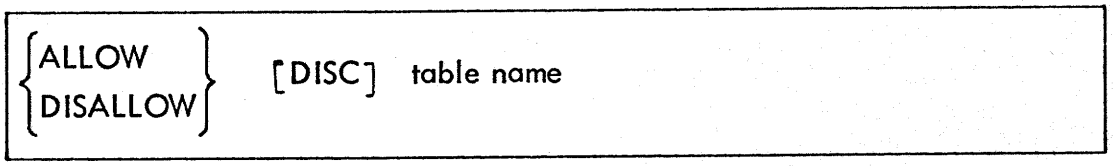
ALARM statement:



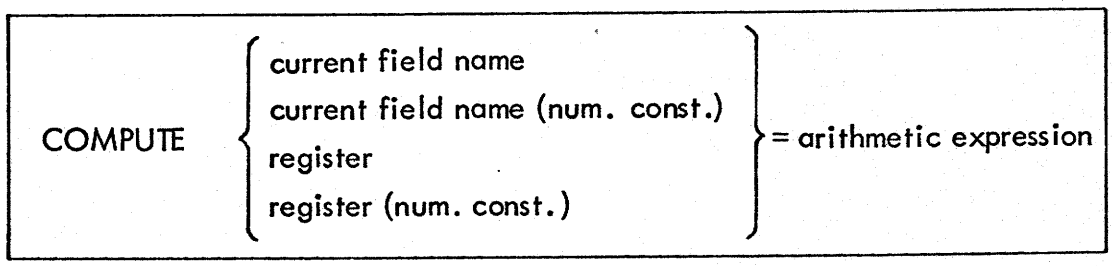
ALLOW and DISALLOW statement:



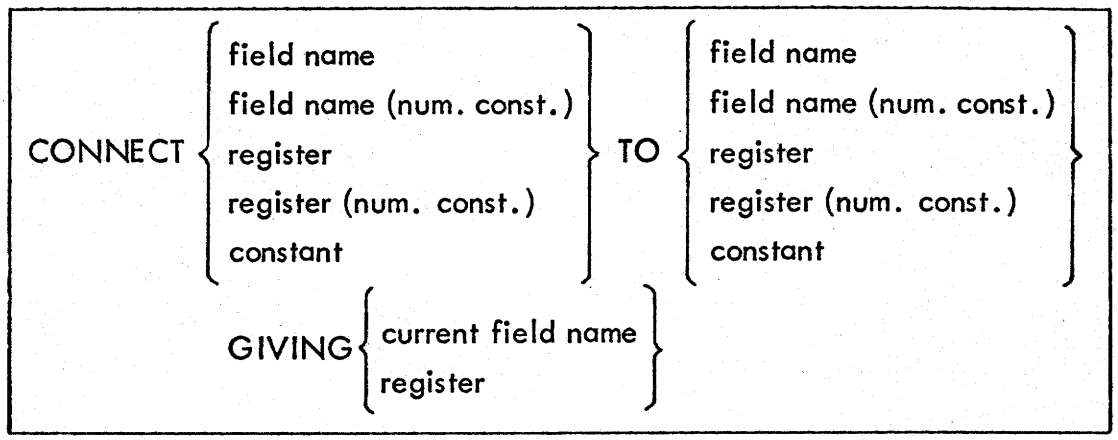
or:



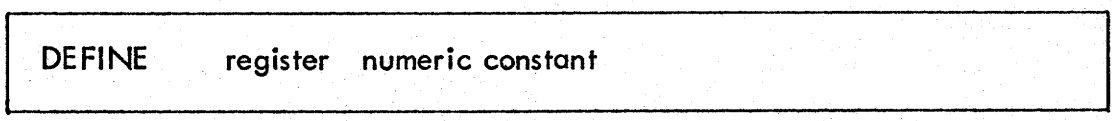
COMPUTE statement:



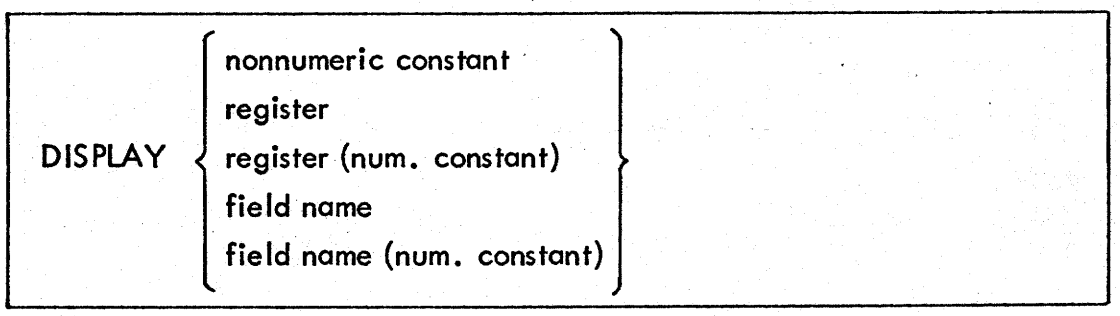
CONNECT statement:



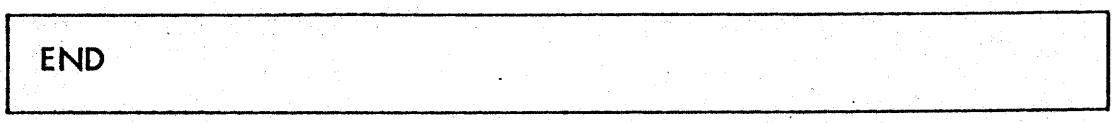
DEFINE statement:



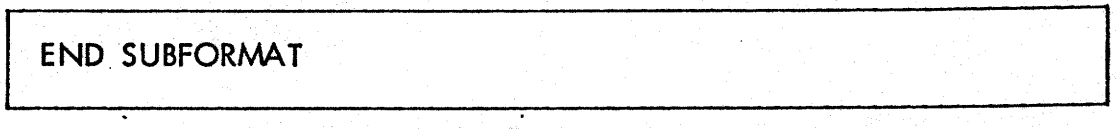
DISPLAY statement:



END statement:



END SUBFORMAT statement:



GOTO statement:

```
GOTO  label
```

LIMIT statement:

```
LIMIT  { -numeric constant }  { -numeric constant }
       { numeric constant  }  { numeric constant  }
```

MOVE statement:

```
MOVE { field name                    } TO { current field name                    }
     { field name (num.const.)       }    { current field name (num.const.)       }
     { nonnumeric constant           }    { register                              }
     { register                      }    { register (num. const.)                }
     { register (num. const.)        }
```

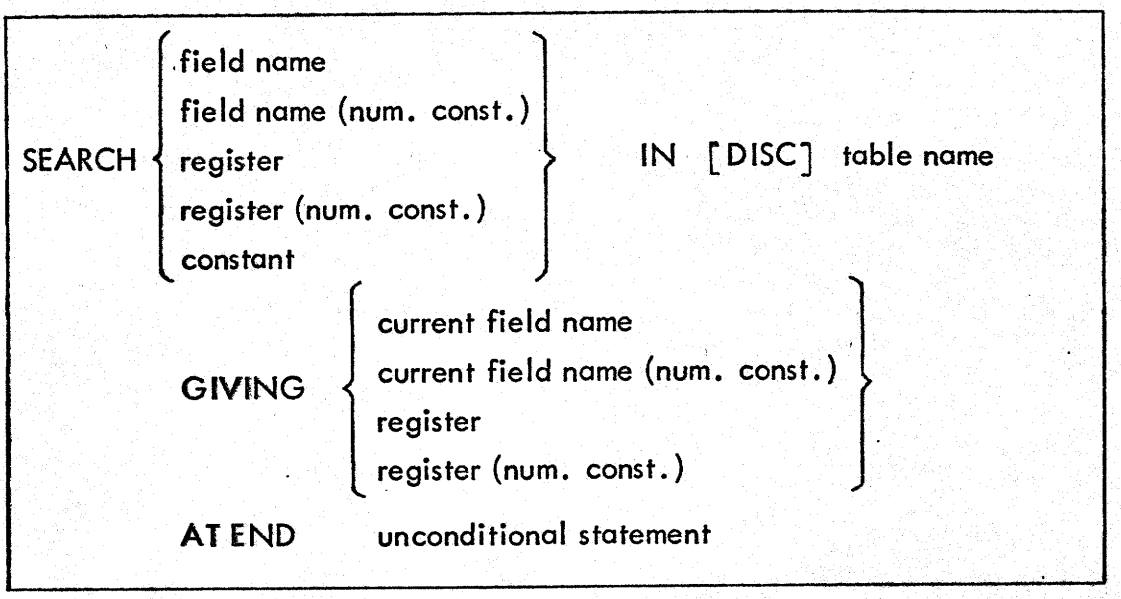
NOTE statement:

```
NOTE  character string
```

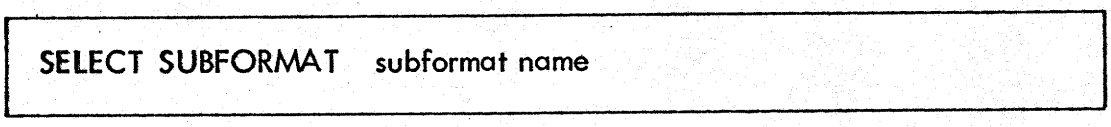
PERFORM statement:

```
PERFORM  subprogram name
```

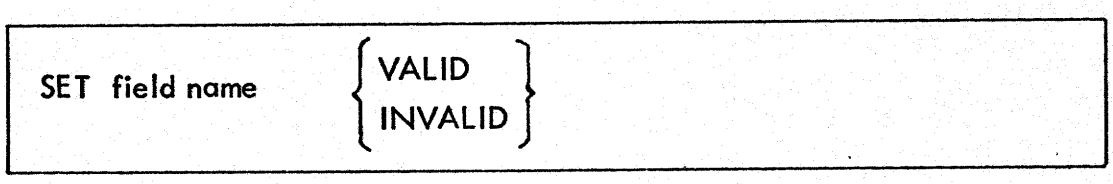
SEARCH statement:



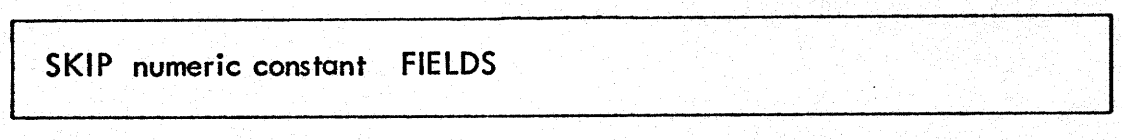
SELECT statement:



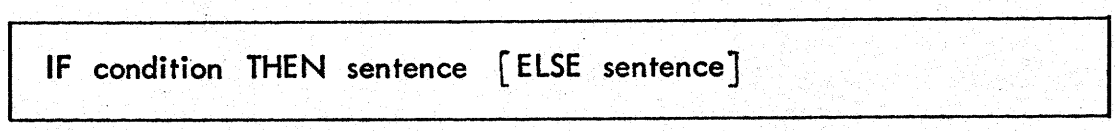
SET statement:



SKIP statement:



IF statement:



APPENDIX VI

Limitations

Maximum number of fields in a subformat	255
Maximum number of elements i 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
Maximum number of formats	84
Maximum number of subprograms	84
Maximum number of tables	84

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

Error Messages From TRANS

All errors are printed on the line printer.

If an error causes a stop in the execution of TRANS 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 TRANS.
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 linenumber 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 SUBPR-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>'th 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-, sub-program-, 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>'th record in the batch is defining a sub-format 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>'th record in the batch is defining the 255th field in subformat <subf>.

8. No field <recordno> <subf> STOP

The <recordno>'th record in the batch is the first one after a subformat - head - record, and contains no check-box (i.e., the field holding field length is empty).

9. String not terminated <recordno> <subf> <fieldno> STOP

The program has observed a text-start-mark with no text-end-mark in the same line; the error is detected in the <recordno>'th record in the batch defining the <fieldno>'th field in subformat <subf>.

10. Illegal formatname <batchname> <recordno> <formatname> STOP

The <recordno>'th record in the batch is defining a new subformat to a format identified by <formatname>.
The batch contains at least two subformats with unlike formatnames.

11. Illegal number of records <batchname> <should have been> <was> STOP

The <was>'th 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.

12. Not in use.

13. Double defined outpos <subf> <fieldno> <outpos> SKIP

<outpos>'th output-field number has been defined twice in subformat <subf>; the error is detected at the <fieldno>'th field.

14a. Illegal statement type <subf> <fieldno> -/SKIP

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.
An end-, endsubformat-, or select-statement is detected in <fieldno>'th field in subformat <subf>, but this field is not the last one in the subformat, 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

TRANS 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>'th 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> SKIP
The program has observed an expression before 'IN' in a table condition.
Concerning parameters, see above.

28. Illegal registerno <subf><fieldno><lineno><symbolno> SKIP
The program has found a reference to register zero.
Concerning parameters, see above.

29. Illegal index value <subf><fieldno><lineno><symbolno> SKIP
A subscript has to be greater than zero and less than 256.
Concerning parameters, see above.

30. Too many items <subf><fieldno><lineno><symbolno> SKIP
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.
Concerning parameters, see above.

31. Illegal recstatus <batchname><recordno><recstatus> STOP
The <recordno>'th record contains an input error.

32 Illegal format structure <batchname> STOP
The batch is not terminated by a record created by sub-format E.

33. Nanny does not exist STOP
The nanny (data entry) system is gone, please fill in an error report for Regnecentralen and restart the system.

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 another 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>'th record in the batch, this record is defining the <entryno>'th 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>'th 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 number 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>'th 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 positions are allocated to <line>'th line in the <page>'th page of subformat <subf>.

45. Disc tables not allowed in subprograms <subf> <fieldno> <lineno> <symbolno> SKIP
References to DISC tables must not occur in subprograms.
Concerning parameters, see above.

46. Registername <> tablename <subf> <fieldno> <lineno> <symbolno> <registername> SKIP
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. <code> may be one of the following ones:

Code	Meaning
001000	The discfile is reserved by another system function. Wait for this function to complete and reenter the command.
004010	Too many programs using the same discfile simultaneously (should not occur, software error).
004020	The program tries to access a disc block outside the file (normally end of discfile reached during reading). Or the program is trying to access a file without being the user of it (software error).
004400	No more resources (area processes) for communicating with the disc. Wait for one or more terminals to end keying and reenter the command.
005000	A number of reasons may cause this status: - An illegal operation is executed due to a software error. - The program tries to reserve a file already reserved by another system function. Wait for this function to complete and reenter the command. - Too many programs using the same disc file simultaneously (software error). - The program tries to access a disc block outside the file due to a software error.
010010	It is not possible to create a file either because the specified file length is too large or because the disc is full (map full) in which case batches no longer needed in the system should be deleted.
010020	It is not possible to create a file as a file with the same name already exists.
010400	No more room on the disc. Batches no longer needed in the system should be deleted.
011000	It is not possible to remove or rename a file as it is presently being used by another system function. Wait for this function to complete and reenter the command.

The statusword may also mean that the system tries to carry out an illegal operation due to a software error.

044000, 050000 — The specified name does not exist as a discfile.

104000, 110000 — Catalog I/O error because the disc is disconnected, because the wrong disc has been mounted, or because of disc malfunction.

When TRANS has finished, SUPV will print

ok	which means	successful translation, the format occurs in library,
cf list	which means	look for error printouts at the line printer,
printer <results>	which means	hard error on the line printer,
break <result>	which means	that TRANS has been aborted by the system.

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
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

Index

Numbers referring to figures are underlined.

READER'S COMMENTS

Data Entry System
RCSL 42 - i 0395

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:

Address:

Organization:

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