DATAFORM 2
CASSETTE DATAFORM 2
User's Guide

January 22, 1975

Model Code No. 50104

DATAPOINT CORPORATION

The Leader in
Dispersed Data Processing

DATAFORM USER´S GUIDE

SECTION I

OPERATION

INTRODUCTION

DATAFORM provides a personalized data entry system for use on a Datapoint 2200. 'Forms', which are kept on tape, project images on the 2200 screen. The data entry operator then simply fills in the form, the data will be recorded on cassette and may at any time be retrieved and revised using the same form to view and edit the recorded data.

Each 'form' is designed by the user, and editing criteria are assigned to the data fields on the form at the time the form is generated. Field programs may also be assigned at this time. The user's forms, and programs, are then combined on a single cassette and become a unique Dataform System configured for the users special purpose.

Four stages of development are involved in generating the system: the Editor/Compiler enables the user to create field programs; the Form Generator enables the user to create forms; the Configurator combines the forms into a 'system'; and the Data Entry Interpreter uses the form to control data entry.

Since DATAFORM uses standardized data tape formats, further processing of the data can proceed under any DATABUS, BASIC or RPG program, or any one of a number of available communications programs or terminal emulators.

TABLE OF CONTENTS
SECTION I

1.0 System Overview	
1.1 What is a FORM?	3
1.2 What is a FIELD PROGRAM?	4
1.3 User space and how it's allocated	5
1.4 Some DATA ENTRY features	6
2.0 Form Generation	
2.1 Data Field	7
2.2 Keyin Field	8
2.3 Data Record	8
2.4 User Space	9
2.5 Form Worksheet	9
2.6 Generating a Form Image	10
2.6.1 Repeat Key	10
2.6.2 Cursor Movement Keys	10
2.6.3 Character Insert Key	10
2.6.4 Character Remove Key	11
2.6.5 Erase Function Keys	11
2.6.6 Line Insert Key	11
2.6.7 Character Duplication Key	11
2.6.8 Returning Control to the Monitor	11
2.7 Assigning Edit TYPES	12
2.7.1 Alpha	13
2.7.2 Numeric Digits	13
2.7.3 Numeric Fields	13
2.7.4 Mixed	13
2.7.5 Left Justified and Zero Fill	13
2.7.6 Right Justified and Zero Fill	13
2.7.7 Right Justified and Blank Fill	13
2.8 Assigning Keyin Restrictions	14
2.8.1 Required Field	14
2.8.2 Fill Control Field	14
2.8.3 Required and Fill Controlled	14
2.8.4 Program Reserved - no keyin	14
2.8.5 Required and Program Reserved	15
2.9 Constants and Semi-Constants	15
2.10 Field Program Assignment	16
2.11 Form Linking	16
2.11.1 Setting Manual Linkage	16
2.11.2 Clearing Linkage	17
2.11.3 Loading Linked Forms	17
2.11.4 Setting Auto-Linking	17
2.12 Writing the form to tape	19
2.13 Changing the screen image	20
2.14 Modifying an OLD form	20
3.0 Form Library Configuration and Utilities	
3.1 Displaying the Form Catalog	21
3.2 Adding New Forms to the Catalog	21

3.3 Deleting Forms from the Catalog 21
3.4 Deleting Multiple Forms from the Catalog 21
3.5 Replacing Forms already in the Catalog 22
3.6 Running the Interpreter 22
3.7 Tape Utility Features 22
3.7.1 Duplicating DF2SYS 22
3.7.2 Duplicating DF2SYS with Forms 22
3.7.3 Recovering a Cataloged Form 22
3.7.4 Generating a Faster Loading Interpreter 22
3.7.5 Copying a Data Tape 22
3.8 Printing Forms and Data 23
3.9 Replacing Dataform2 Systems Programs 24

4.0 Data Entry Interpreter
4.1 Data Tape Initiation 25
4.1.1 START new data tape 25
4.1.2 ADD to an existing data tape 27
4.1.3 CONTINUE an existing data tape 27
4.2 Loading a Form 27
4.2.1 Testing NEW Forms 27
4.2.2 LOADing Cataloged forms 27
4.3 Entering DATA Entry Mode 28
4.4 Revising Exisiting Data Tape 28
4.4.1 MODIFY 28
4.4.2 Rewriting Existing Records 29
4.4.3 FINDing a Specified Record 29
4.4.4 Aborting Tape Searches 30
4.5 Positioning the Data Tape 30
4.5.1 BACKUP tape when entering data 30
4.5.2 REWINDing during modification 30
4.6 ENDing the Data Tape 31
4.7 Data Entry Action 31
4.8 Data Erase Function Key 32
4.9 Field Duplication Function Key 32
4.10 Operator Errors During Data Entry 32
4.11 Write Function Key 32
4.12 Loading Linked Forms 33
4.13 Returning Control to the Monitor 33
5.0 Recovery Proceedures
5.1 System Tapes 34
5.2 Data Tapes 34
5.3 Form Tapes 35

APPENDIX A
TYPE and REQUIRE codes

APPENDIX B
Form Generation Function Keys

APPENDIX C
Command List

APPENDIX D

Data Entry Function Keys

APPENDIX E
Error Messages

APPENDIX F
Examples of Form Generation

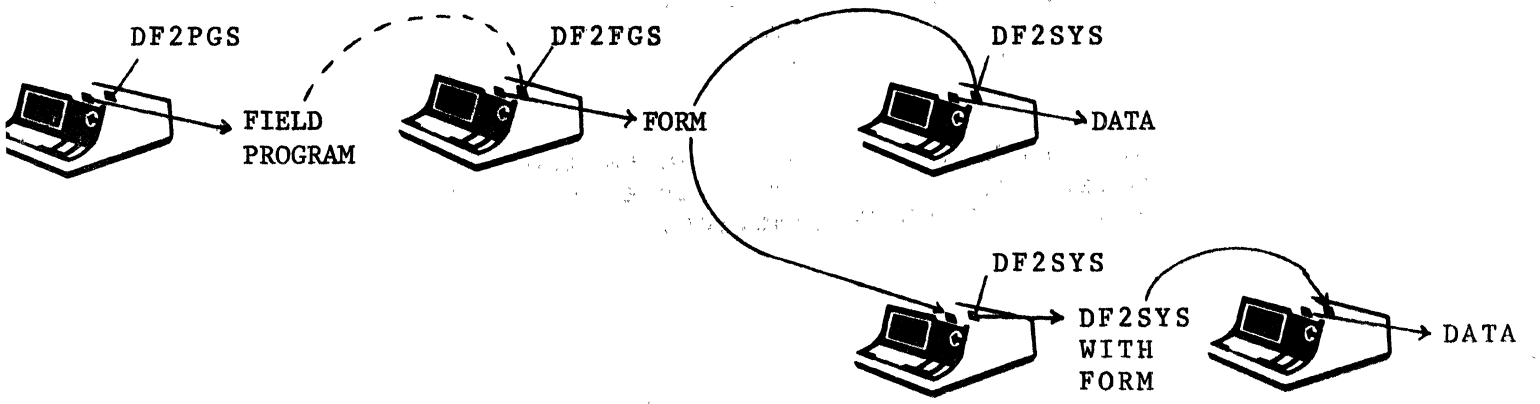
DATAFORM 2 SYSTEM OVERVIEW

The DATAFORM 2 System is composed of three separate system tapes:

1. DF2SYS, Data Entry System tape, contains the Configurator, the Form Catalog and the Data Entry Interpreter. The Configurator enables the user to add forms to this system tape for use during data entry.

2. DF2FGS, Form Generation System tape, includes the Form Generator, the Field Program Relocator and 15 versions of the Extended Interpreter.

3. DF2PGS, Program Generation System tape, consists of a source code editor and the DATAFORM Compiler.


DF2PGS
FIELD PROGRAM
DF2FGS
FORM
DF2SYS
DATA
DF2SYS
DF2SYS WITH FORM
DATA

SYSTEM FLOW

1.1 What is a FORM?

A ´form´ in this document refers to a screen image designed using the DATAFORM Form Generator. This screen image contains labelling information, defines the length and positions of ´data fields´, and reserves space for ´keyin fields´.

The amount of data, the number of fields and the amount of constant information in the form image determine exactly how much memory the form requires.

The Form Generator also enables the user to assign edit criteria to the data fields. The criteria are applied field-by-field in separate passes over the form image.

These criteria include the field type:

> alpha,
> numeric digits,
> alpha/numeric,
> numeric left justified/blank filled,
> numeric left justified/zero filled,
> numeric right justified/zero filled,
> numeric right justified/blank filled;

entry restrictions:

> required,
> fill-controlled,
> required and fill-controlled,
> no keyin,
> required and no keyin;

constant data, semi-constant data, and automatic form control (linking to other forms).

In addition, ´field programs´ may be assigned during form generation. Up to twenty-six unique field programs may be referenced in a single form.

If a field program is specified, the user must have prepared the program previously (edited and compiled using DF2PGS). At form generation time all referenced field programs are sought and combined with the form into an object format called the ´form tape´.

The screen image, basic edit criteria and field programs, if any, comprise the ´form´ which is subsequently executed by the DATAFORM Interpreter.

1.2 What is a FIELD PROGRAM?

If editing is required beyond that available in a basic form, the user may write a program in the DATAFORM language. This language provides the user access to the entire data record (on a character or field basis) and the ability to define working storage variables, tables, messages, etc. One hundred bytes of common storage is available to pass information between forms. The DATAFORM language provides the user with the following editing capabilities:

- Arithmetic
  - Add
  - Subtract
  - *Multiply
  - *Divide
- Data Manipulation
  - Move
  - Align
  - Set
  - *Lookup
  - *Convert
- Table Checking
  - In-table
  - Not-in-table
  - In-range
  - Not-in-range
- Check Digits
  - *Mod10
  - *Mod11
- Compares
  - Less than
  - Greater than
  - Equal
  - Not equal
  - Less-than-or-equal
  - Greater-than-or-equal
  - Not-equal
- Branching
  - Go to
  - Call
  - Return
  - Again
  - Next
  - Store
  - Change
  - Reset

Input/Output
- Write
- Close
- Message
- Show
- Beep
- End
- Load

The subroutines to execute these commands are divided into two groups: the Interpreter and the Extended Interpreter. The starred (*) commands in the preceeding list require the Extended Interpreter. (See the discussion of user space below.)

Field programs are entered using the Editor on DF2PGS. This creates a ´source´ tape. When editing is completed, the Compiler automatically compiles the DATAFORM statements and adds a ´relocatable object´ file to the end of the source file.

The field programs will be assigned, in a pass of the Form Generator, to particular fields. When the form is written out, the relocatable program will be converted to ´absolute´ code and written to the form tape. If, in addition, an extended interpreter is required by a particular field program, the required file is copied from the DF2FGS tape to the form tape.

During data entry (using DF2SYS) the field program is executed after the operator enters data into the field where the program assignment was made. The program is executed even if the operator bypasses the field.

1.3 User space and how it´s allocated

When a new form is being created, there is 1550 bytes of memory available to the user. This ´space´ must eventually contain:

- keyin data buffer
- writing data buffer
- form image
- user programs (if required)
- extended interpreter (if required)
- common storage

The Form Generator indicates the amount of free space as soon as the form image has been defined. The user must then determine if his program and, if necessary, the Extended Interpreter will fit in the remaining space.

1.4 Some DATA ENTRY features

Special keys are available which provide the operator with the following functions:

- field duplication
- form data erase
- data write
- forward field tab
- backward field tab
- field cancel
- form load
- record read
- record backspace

Semi-constant data may be defined in the form and simply accepted by the data entry operator.

Forms may be loaded in non-sequential order under either program or operator control.

Operator correction of previously generated data may be accomplished at any time by either a manual, record-by-record, or an automatic search with re-writting in-place permitted.

Data may be added to the end of an existing data tape (positioning is automatic).

2. FORM GENERATION

A Dataform ′form′ is an image projected on the 2200 screen which contains <u>form text</u> (explanatory information for the operator, not to be written to the data tape), <u>field definitions</u> (special characters which define an area to be filled in by the operator and to be written on the data tape) and <u>keyin space</u> (special characters which define an area to be typed into [but not stored in the data record]). The 2200 screen is 80 characters wide and 12 lines high and any of the 960 positions on the screen may be used in the form.

2.1 Data Field

A data field is part of the form image which starts at a vertical bar (|) and is continued by carets (^) or underscores (_). A field stops at the first non-caret, non-underscore or the right hand edge of the screen.

Each data field causes a corresponding number of positions to be reserved in the two data areas (one for keying in and one used for writing) and each field generates a six byte edit table entry. Each field defined has a ′field number′ corresponding to its relative position in the form (and pointing to it′s entry in the edit table). The uppermost, leftmost field is number one. Fields are numbered from left to right, line by line, from the top of the form down.

This construction, ′|^^^′, defines a four position data field; ′|′ defines a single position field and ′|||′ defines three adjacent single position fields. The differences between one 3-position field and three 1-position fields are:

> 1) only one edit criteria applies to the 3-position field whereas each 1-position field may be assigned a different edit criteria;
>
> 2) since each additional field definition takes 6 bytes, the three 1-position fields use more user space than the single 3-position field.

Fields defined by carets will be ′space compressed′ in the form image. This means that the carets will be replaced by spaces which will, in turn, be replaced by a space compression character (011) and a count of the number of spaces compressed. (The count will include any spaces trailing the field.) When the form is displayed, space compressed fields will initially appear blank. As the cursor enters the field, the appropriate number of underscores will be displayed.

Fields defined by underscores are not compressed. The underscore characters are saved as part of the form image.

Constants and semi-constants are stored in the field description area of the form image and therefore, can be defined only for fields initially defined by underscores.

The maximum number of characters in a single data field is 80 since the <u>right hand edge of the screen always terminates the field definition</u>.

2.2 Keyin Fields

A keyin field, with the exception of the initial character, is defined exactly as is the data field. Keyin fields begin with a less than character (<) and are continued by carets or underscores. They may appear anywhere in the form. Keyin fields create a six byte entry in the field edit table and thus have a corresponding 'field number'. However, no space is reserved for these fields in the data record.

2.3 Data Record

The length of the data record generated during data entry is determined by the combined lengths of all data fields in the form (maximum 245 characters). The data record will also automatically contain a form number (1 binary character) and a rewrite counter telling the system how many times it was corrected (1 character). The format of the record is:

data fields (written to their defined lengths) in the order they appear on the form (from left to right and from top to bottom)

logical and physical record terminators (015,003)

form number (1 binary character) which corresponds to physical file number of the form

rewrite counter (1 ASCII character)

2.4 User Space

There is a fixed amount of space available in which to contain the form image, the data input/output areas, the field edit tables and field programs. This variable area is called <u>user space</u>. There is no limit (other than the size of the screen) to the amount of text one may include in a form. There is, however, a limit to the number of field definitions (126) and to the number of data characters (245) which can be defined. The total 'user space' available is 1550 bytes.

The number of data characters, defined in the form image, reserve two areas: the keyin data area and the writing data area. In addition, each field (whether an actual data field or a keyin only field) defined in the form image requires a six byte edit table entry. The characters displayed in the form image, both labelling information and field defining characters (excluding carets) reserve user space. Spaces (and carets) in the form image are 'compressed', i.e., they are represented by a space compression character followed by the number of spaces compressed at that point. One terminator character is added to each line of the form image; however, lines which are completely blank require no space at all.

The amount of user space reserved for the data record, edit table and form image is subtracted from the total user space and the amount remaining is indicated at the end of the form image generation pass.

In addition to the data record, edit table and form image, the user may allocate user space to field programs (which in turn may require an extended interpreter). The length of a field program is indicated on the listing and on the CRT at the end of program compilation. If an extended interpreter is required, the user should refer to the size table in Section II, Appendix C.

When the form tape is generated (by the OUT command), the amount of user space remaining (or the excess allocated, if any) is indicated.

2.5 Form Worksheet

To aid the user in designing his forms, a 'Dataform Worksheet' is available. This worksheet provides space for designing the screen image and for recording the various criteria, constants, etc. which will have to be assigned at form generation time. The worksheet also serves as a record of the form and as a quick reference for Generator commands and function keys.

A printout of completed forms, similar in format to the worksheet, may be obtained using the print utility of the Configurator. (See Appendix F.)

2.6 Generating a Form Image

To generate a new form, load DF2FGS and type NEW to clear the screen and switch to the image generation mode.

When the DISPLAY key is pressed, the number pad, to the right of the keyboard, or the regular number keys become a set of special function keys enabling the user to move the cursor up, down, left and right, to insert and delete characters, to delete words, to insert lines and to erase lines and portions of the screen.

A key becomes a function key if it is pressed simultaneously with the DISPLAY key, i.e., first press the DISPLAY key and, while holding it, press the key required to achieve the desired function.

2.6.1 Repeat Key

Holding the KEYBOARD key will cause the character (and many functions) to be repeated. Press the KEYBOARD key first, then, if a function key is needed, the DISPLAY key, then the desired character.

2.6.2 Cursor Movement Keys

There are five cursor movement keys which are non-destructive, i.e. they pass over characters on the screen without erasing them. The number two function key moves the cursor DOWN, the number eight function key moves the cursor UP, the number six function key moves the cursor to the RIGHT and the number four function key moves it to the LEFT.

The BACKSPACE key also moves the cursor to the LEFT in a non-destructive manner. Backspacing will wrap around from column 1 of a line to column 80 of the preceeding line.

The SPACE bar is destructive, i.e., it erases the characters it passes over, and moves the cursor to the RIGHT.

All cursor movement keys may be repeated.

2.6.3 Character Insert Key

The number seven function key, at the upper right of the number pad, causes a space to be inserted in the line on the screen. This key may be repeated. Characters at the end of the line are truncated, not wrapped around.

2.6.4 Character Remove Key

The number zero function key, at the lower right of the number pad, causes the character under the cursor to be removed and the remaining characters to be concatenated to the left. Spaces are set in the trailing positions.

2.6.5 Erase Function Keys

There are several keys available to erase all or part of the screen image. The word remove key, number one function key, causes a word, a group of characters edged by spaces, to be removed and the line to be concatenated.

The period function key, erase end of line, causes the line to be erased from the cursor to the right hand edge of the screen.

The number nine function key, erase end of frame, causes all characters to be erased from the cursor to the end of the screen, i.e., through line 12 character 80. This key can easily clear the entire screen.

The CANCEL key causes the entire line that the cursor is resting on to be erased.

2.6.6 Form Expand Key

The number three function key causes a blank line to be inserted at the line where the cursor is resting. The line under the cursor and all subsequent lines are moved down the screen. If there is anything on the twelfth line, it will disappear.

2.6.7 Character Duplication Key

The number five function key causes the character immediately above the cursor to be duplicated in the current cursor position. This function key may be repeated. (It has no effect on the top line of the screen.)

2.6.8 Return to Monitor

When the screen has the desired appearance, press the escape function (DISPLAY/CANCEL) to get back to the Generator's monitor. At this point a message will appear:

```
nnn DATA

mmm BYTES LEFT
```

informing the user of the number of characters in the data record and of the number of bytes remaining in the user space. If the number of characters is greater than 245, the message:

```
MORE THAN 245 DATA
```

will appear. The form must immediately be revised to reduce the number of characters. If more than 126 fields are defined, an error message:

```
MORE THAN 126 FIELDS
```

appears. Again, the form should be immediately revised to reduce the number of fields.

If the combined space required by the form image, data areas and edit tables exceeds the available user space, the message:

```
nnn BYTES OVER
```

will appear. The form should be revised to fit the user space available. (See Appendix D in Section II for suggestions on saving space.)

The form is still only in memory and no edit criteria have been assigned.

A ´pass´ is a <u>pass over the form to assign edit criteria</u> to it. It is up to the user to request each type of edit-defining pass (TYPE, REQUIRED, PROGRAM, CONSTANT, SEMI-CONSTANT, LINK) and, finally, to record the form by use of the OUT command. The edit-defining passes may be requested in any order. Any or all edit-defining passes may be omitted, and passes may be repeated to review or to change the criteria.

2.7 Assigning Edit TYPES

Generally, once the form image has be created, the user will first assign edit types by keying the ´TYPE´ command. This command will initiate the TYPE pass over the form causing the form to be redisplayed with the cursor at the first field definition (i.e., the first vertical bar [¦] or less than sign [<]). The user may then type one of the legal edit types (A,D,N,M,L,R,B), or press the ENTER key to pass the field without changing the criteria, or press the CANCEL key to clear the previously set criteria (no editing will be performed on a cleared field; however, other passes may still be executed to set restrictions such as required or field program execution).

If this pass is re-executed, the current edit types will be displayed as each field is reached. If no change is needed, just press the ENTER key to proceed.

The 'B' function key may be pressed to position back to the previous field. When the desired edit types have been assigned, the escape function key (DISPLAY/CANCEL) will return control to the monitor.

2.7.1 Alpha

The 'A' edit type indicates the characters keyed in must be uppercase alphabetics (A through Z) or space. The field will be space filled to the right if not completely keyed in.

2.7.2 Numeric Digit

The 'D' edit type indicates the characters keyed in must be strictly numeric (0-9). The field is left justified with trailing blank filler.

2.7.3 Numeric Field

The 'N' edit type indicates the characters keyed in must be numeric (0-9), a decimal point or a minus sign (plus signs are <u>not</u> allowed). Numeric fields are left justified and blank filled on the right.

When the field is entered, it is checked to contain one decimal point at most. If a minus sign is present, it must be the left most character. And, no more than twelve positions are permitted to the left and four to the right of the decimal point.

2.7.4 Mixed

The 'M' edit type signifies alphanumeric, i.e. characters A through Z, space, 0 through 9, decimal and minus are permitted. The field is space filled to the right if not completely keyed in.

2.7.5 Left Justify and Zero Fill

The 'L' edit type has the same restrictions as type 'N'; however, the input is left justified and zero filled on the right.

2.7.6 Right Justified and Zero Fill

The 'R' edit type has the same restrictions as type 'N'; however, upon completion of data entry, the input is right justified in the field and zero filled to the left.

2.7.7 Right Justified and Blank Fill

The 'B' edit type has the same restrictions as type 'N'; however, upon completion of data entry, the input is right justified in the field and blank filled to the left.

2.8 Assigning Keyin Restrictions

To establish that a field may not be bypassed (tabbing past without entering data) during data entry, or that all characters must be entered, or that the field is not to be keyed-in but is reserved for a field program, use the REQUIRE command.

This command will cause the form to be displayed with the cursor at the first field. The user may then type one of the options (R,F,B,P,S) to set the appropriate restriction, or press ENTER to go on to the next field, or press CANCEL to clear a previously set required condition.

The currently set condition will be displayed as the field is entered. The ENTER key will tab forward and leave the condition unchanged; the 'B' function key will tab back to the previous field.

2.8.1 Required Field

An 'R' keyed in during this pass indicates a field is required, that is, at least one character must be typed.

2.8.2 Fill Control Field

An 'F' means fill-controlled, that is, all characters must be keyed in. During data entry, fields defined as fill-controlled will be automatically entered when the last character is keyed in; however, the ENTER key may not be pressed in such a field. If fill-control is not set, the ENTER key is required, even if the entire field is keyed in.

Fill-control should not be set in fields with edit types R, B or L where action is taken when the ENTER key is struck.

Fill-control fields may be bypassed if the ENTER key is struck in the first column of the field.

2.8.3 Required and Fill Controlled

The 'B' option sets both required (R) and fill-control (F).

2.8.4 Program Reserved - No Keyin

The 'P' option indicates a field will be filled in by a field program. No operator keyin is permitted in this field.

This option may also be set on a 'keyin' field to reserve it as an alternate message display area.

2.8.5 Required and Program Reserved

The 'S' option sets both program reserved (P) and required (R). This will prevent writing of the data record if the program reserved field has not been set by the field program.

2.9 CONSTANTS and SEMI-CONSTANTS

The user must consider carefully the implications of 'constant' and 'semi-constant' data. These are characters set into a data field in the form image and, if not overridden, they will become part of the data record during data entry. During actual data entry the operator has the option to accept or type over data set by the SEMI-CONSTANT command; whereas, data set by the CONSTANT command will automatically become part of the data record and cannot be rejected by the operator.

Constants and semi-constants may only be set in fields initially defined, i.e. at image generation time, by underscores.

To initiate the constant pass, type 'CON'. To initiate the semi-constant pass, type 'SEM'. Both commands cause the form to be displayed with the cursor in the first field capable of accepting constant type information.

In the constant setup mode, the SPACE bar does not set constant spaces into the field but permits the user to move to a desired position within the data field. If constant spaces are required, the caret key (^) must be used. In addition, neither constant nor semi-constant underscores (_), veritical bars (¦) or carets (^) can be set. The CANCEL key will clear any constant field erroneously set, the ENTER tabs forward, and the DISPLAY/B keys tabs backward. The BACKSPACE key positions back one character and erases the last character typed.

No editing is performed on constants entered at this time. Illegal constants will cause the Interpreter to hang beeping during data entry. Illegal semi-constants, although rejected, will be displayed. This feature may be useful for presenting additional information to the operator, e.g., a date field may have the illegal semi-constant 'YYMMDD' set to guide the operator.

Also, if an entire form of constant data is prepared, at least one position must be left for the operator - so that the form may be viewed and/or written to tape. All-constant forms (or forms with no fields) will cause the Interpreter to <u>hang clicking</u> at data entry time.

Partial semi-constants at the beginning or in the middle of a field are meaningless since the operator will have to type over them to enter the remainder of the field.

Once constants (of either type) have been set, they will always appear when the form is displayed (e.g., during the TYPE pass). Typing over the constants during subsequent edit definition passes will not disturb them.

Constants should be cleared before executing the REVISE command since <u>their presence will change the data field definitions</u>.

2.10 Field Program Assignment

Field Programs are written in the DATAFORM 2 Language (See Section II). Each program is identified by a single alphabetic character (A - Z).

When the form is being generated the user must type 'PRO' to enter the program assignment pass. The form will be displayed with the cursor in the first field. Type the appropriate program letter in any field where special processing is required.

The same field program may be assigned to several fields, e.g., a year and month range check could be used for any date field. Up to twenty-six unique field programs may be assigned in one form.

2.11 Form Linking

Dataform provides the user with the ability to 'link' forms so that the operator need never see a form number. Each form on a Dataform System may have a pointer, called a 'link', to the next form to be used. This pointer must be defined at form generation time. When designing the forms for a particular application, the user should plan the numbers that will be assigned to each form and plan to arrange the forms for the most convenient accessing.

2.11.1 Setting the Manual Linkage

When the LINK command is typed, the message:

'NEXT FORM nnn:'

will appear. (where nnn is the number of the current linked form, initially 000). Type the number of the form to be used by the data entry operator after the form currently being generated has been filled in.

2.11.2 Clearing the Linkage

Typing 'LINK', and entering a zero when the 'next-form' message appears, clears the link so that a form load command issued by the operator will have no effect.

The current linkage information may be viewed by typing the LINK command and then simply pressing the ENTER key to leave the values unchanged.

2.11.3 Loading Linked Forms

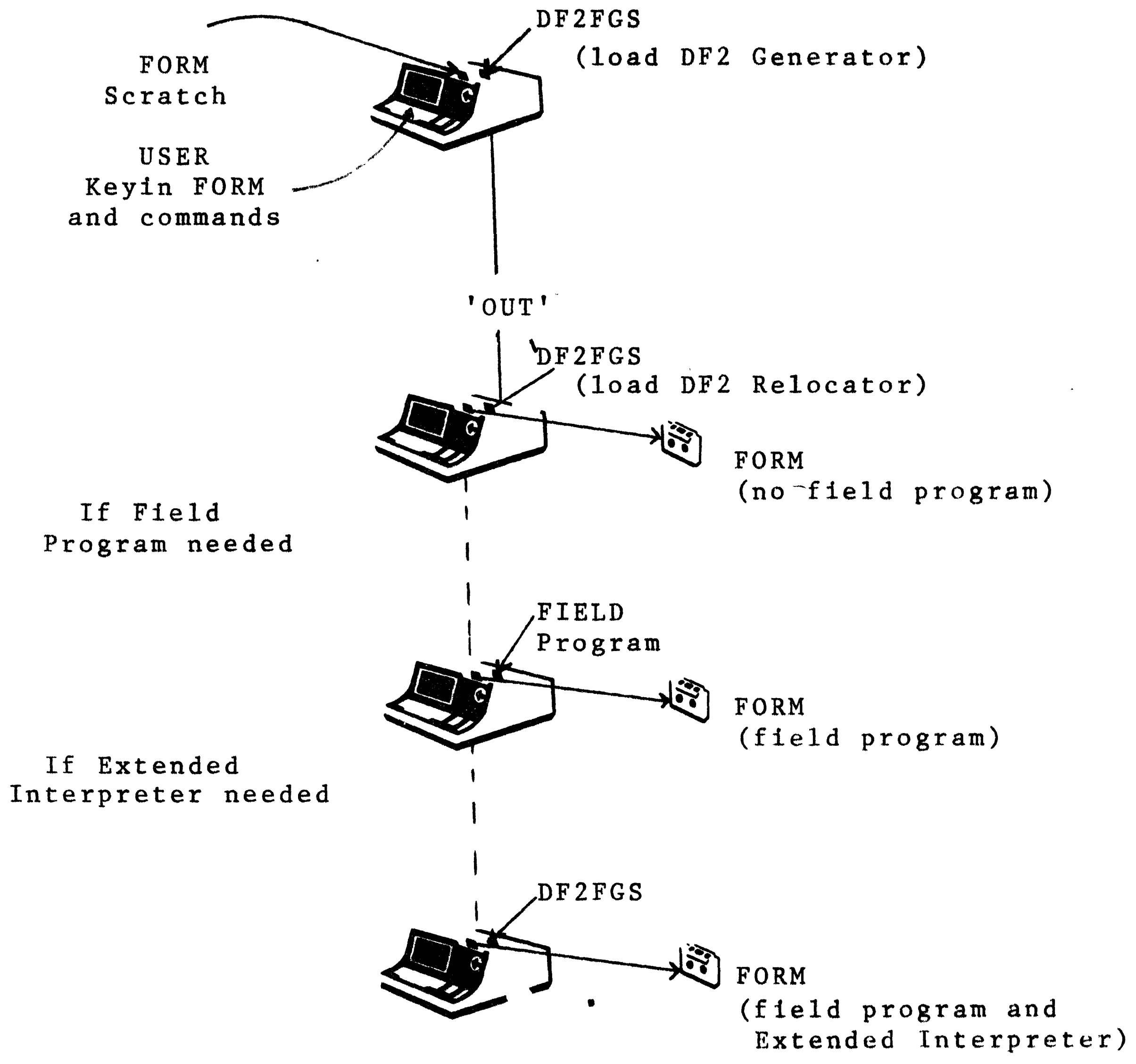
During data entry, the linked form (set at form generation time by the LINK command) is accessed by the operator, when needed, by a special form-load function key.

2.11.4 Setting Auto Linking

One user form may require several Dataform 'forms', e.g. forms 1, 2 and 3 make up one payroll transaction. In order to fill in form 1 once, then form 2 once, then form 3, the operator would have to use the write function (to write out the data) and then the form-load function (to load the next form). (See also Section II - CHAIN and WRITE.)

To facilitate use of mulitiple page forms (i.e. sets of forms to be completed in sequence and then reused), the next form links can be set at form generation time to auto-load new form whenever data is written.

To set the auto-linking feature use the LINK command. When 'NEXT FORM' is requested, preceed the form number with a minus sign. Thus, when generating form one in the multi-page example above, enter '-2' as the next form; enter '-3' for form two and '-1' for form three's auto-link.



FORM GENERATION FLOW

2.12 Writing the Form to Tape

<u>During the entire form generation time the form is only in memory</u>. To record the form and its associated edit criteria, place a scratch tape in the front deck and type the command OUT. If no errors have been detected (e.g. too many fields, too long a data record), the Field Program Relocator will be loaded. If no field programs have been specified, the Relocator will automatically write the 'form tape' on the front deck. If programs have been specified, the question:

```
DO YOU HAVE A PROGRAM TAPE?
```

will appear. If the field program is not yet prepared, type 'NO'. This will cause all references to the program to be ignored during data entry.

If, however, a new program is to be used and the operator types 'YES', the message:

```
PLACE TAPE IN REAR DECK AND PRESS ENTER
```

will appear. Remove the DF2FGS tape and place the field program tape (source/object created by DF2PGS) in the rear deck and press ENTER. The program tape will be rewound and searched for the required programs, which will be copied to the front deck. If there are still unresolved program references, the program-tape message will be repeated. When all programs have been supplied or the user has answered 'NO' to the program-tape message, the form is written on the front deck. Then the 'extended interpreter flag', set by the compiler for each of the field programs, is examined to determine which, if any, of the extended interpreter functions are required. If no extended interpreter is needed, the form tape is ended. However, if the field programs require an extended interpreter, the message:

```
REPLACE DF2FGS AND PRESS ENTER
```

will appear. The user should put the Form Generation System tape back in the rear deck and press ENTER. The appropriate Extended Interpreter (containing only the commands executed by the field programs) will then be copied onto the end of the form tape thus far generated.

At the completion of the form writing process, the message:

```
nnn BYTES LEFT

DONE - LOAD NEXT SYSTEM
```

will be displayed and the machine is stopped. If, instead, the message:

```
     nnn BYTES OVER
```

appears, the user must revise his form or field programs to fit within the available user space.

The form recorded on the front deck may now be tested, by loading the Interpreter and typing NEW, or it may be cataloged on the Interpreter System tape.

2.13 Changing the Screen Image of the Current Form

If, during one of the passes subsequent to image generation, an error in the image of the form is discovered, the user may type the:

```
     REVISE
```

command to return to the image generation mode with the current form intact. <u>All edit criteria are cleared</u> which means that <u>all passes executed will have to be done again</u> after the form has been revised.

If constants had already been set into the form, it is best to go into CONSTANT mode and <u>clear (using the CANCEL key) all constant fields</u> (since constants destroy the field definition characters) <u>before executing the REVISE command</u>.

2.14 Recovering an OLD Form

Once a form has been recorded it may be retrieved and modified. Typing the:

```
     OLD
```

command will bring the form from the front deck into memory. Any pass of the Generator may be executed; however, note that the REVISE command will clear all edit criteria.

When the desired changes have been made, the form in memory is written to the <u>front deck</u> (using the OUT command) as if it were a new form, i.e. all field programs will have to be re-attached.

If the field program associated with a form have changed, simply type OLD, to reload the form, and OUT, to attach the new version of the program. Any time a form is read via the OLD command, all field programs required must be reloaded.

3. Form Library Configuration and Utilities

Once the forms have been generated and tested (see Section I, part 4, Data Entry), the user will wish to catalog them on a Dataform System tape. The system of forms should be designed carefully to provide both the simplest and fastest operation for data entry. The Configurator enables the user to record and manipulate the forms on a Dataform System.

To load the Configurator, place the Dataform System tape in the rear deck and press RESTART. The Configurator is the first program on the tape. However, the system will go automatically to the Interpreter. To override automatic loading of the Interpreter, the operator must press the DISPLAY key during the entire loading process.

3.1 Displaying the Form Catalog

A 'catalog' is maintained on the system tape which identifies the forms available on that particular tape. Each form is identified by a number (1 to 123 are valid form numbers). To see which form numbers have been assigned, type 'CAT'. This command will display the form numbers in use.

3.2 Adding New Forms to the System Catalog

To input a form created by the form generator and assign it a form number, type 'IN nnn', where nnn is the form number being assigned. The command causes the form in the front deck to be input, in form number sequence, to the system tape, and then adds the new form number to the catalog, rewriting the catalog record also. If errors are encountered while the form is being copied, the catalog will not be rewritten.

3.3 Deleting Forms from the System Catalog

Forms may be removed from the system tape, and from the catalog, by use of the delete command (DEL nnn). If the form being deleted is not the last one (i.e., highest number) in the catalog, a scratch tape will be requested for the front deck.

3.4 Deleting Multiple Forms from the System Catalog

There is also a command which allows the removal of multiple forms. The 'CHOP nnn' command deletes the specified form and all subsequent (higher numbered) forms from the system tape and from the catalog.

3.5 Replacing Forms on the System Catalog

If changes have been made to a form already cataloged, it may be replaced (REP nnn). If the form being replaced is not the last one on the system tape, the REPLACE command copies all subsequent forms onto the end of the form tape in the front deck, and then copies the front tape back to the system tape, writing over the specified form.

3.6 Running the Interpreter

In order to proceed from the Configurator to the Interpreter, one may, as mentioned above, simply press RESTART which will cause automatic loading of the Interpreter. There is also a command available (INT) to cause loading of the Interpreter.

3.7 Tape Utility Features

3.7.1 Duplicating DF2SYS

In addition to building and maintaining the forms catalog, the Configurator has several utility commands to assist the user in protecting his system tape. The duplicate (DUP) command generates a Dataform System on the front deck. The new system has no forms in its catalog.

3.7.2 Duplicating DF2SYS with Forms

Another command (DUP-ALL) copies the entire system tape onto the tape in the front deck.

3.7.3 Recovering a Cataloged Form

A single form may be transferred from the system tape to a scratch tape by using the 'OUT nnn' command. The OUT'ed tape then looks exactly like a tape written by the Form Generator and may be IN'ed to another Dataform System or changed using the Generator (the OLD command).

3.7.4 Generating a Faster Loading Interpreter

The LGO command will make a faster loading version of the Interpreter System and its forms by omitting the Configurator. No form manipulation (IN, OUT, REP, etc.) can be performed on the LGO version of the system.

3.7.5 Copying a Data Tape

Data tapes may incur tape parity errors or particular data records may reach the rewrite limit by being modified the maximum number of times. A COPY command in the Configurator enables the user to <u>copy the data</u> tape, resetting the rewrite counter in each record back to zero, and, if tape errors are encountered, provides the option of omitting the record, terminating the copy, or attempting to copy the bad data.

When the COPY command is executed, the message:

```
PLACE DATA TAPE IN FRONT DECK, BLANK TAPE IN REAR DECK
WHEN TAPE IN PLACE, PRESS ENTER
```

will appear. Once the ENTER key is pressed, the tape in the front deck will be copied to the rear deck. If errors are encountered on the master data tape, the following message will appear:

```
PARITY ERROR ON DECK 2
COPY, OMIT OR END?
```

If 'O' is typed, the bad record is bypassed and the copy proceeds. If 'E' is typed, the copy is terminated with end of file markers written on the copy tape. If 'C' is typed, the bad record will be written on the copy tape (the copied record will have <u>no</u> tape error; however, it will probably be missing data or contain erroneous data) and the copy will continue.

If end of tape is reached and no end of file detected, the COPY command will automatically backspace twice and write end of file markers on the copy tape. The master tape is not disturbed. Note that if this occurs, the final record count is unreliable.

When the copy is completed, the following message is displayed:

```
nnn RECORDS COPIED - REPLACE SYSTEM TAPE IN REAR DECK
WHEN TAPE IN PLACE, PRESS ENTER
```

The DF2SYS tape must then be replaced and the ENTER key pressed. The forms catalog will be reloaded.

3.8 Printing Forms and Data

Two utility commands are available to print, either data or forms, on a local or servo printer. To print a data tape, type 'DPRINT'. This rewinds the data tape in the front deck and prints each record, 80 characters per line, on whichever printer is available.

Forms will be printed twice; once as the total image would appear to the operator and again, one line at a time, followed by the size of the field, the edit TYPE, REQUIRED, and PROGRAM codes entered.

To print a non-cataloged form, place the tape in the front deck, type ′FPRINT′ and press ENTER.

To print cataloged forms type:

```
FPRINT nnn
```

where nnn is the number of the form to be printed. To print all cataloged forms, type:

```
FPRINT ALL
```

This will cause all cataloged forms to be printed in the order they occur on the system tape. When printing is completed, the Configurator is reloaded.

3.9 Replacing Dataform2 Systems Programs

One final command is available to replace the entire catalog and forms of one Dataform System with the catalog and forms of another Dataform System. The RIN command is intended mainly for use in upgrading the users system as new versions of Dataform are released.

Place the <u>new DF2SYS in the rear deck</u> and the system tape containing forms in the front deck.

4. DATA ENTRY

Data entry involves loading the Interpreter, then loading a form, and finally filling in the data fields defined by the form. When the data has been entered on the screen to the operator's satisfaction, the data record is written to tape (by an operator function key or a field program instruction) then the same form is cleared and redisplayed with only 'constant' type data appearing.

To start data entry, place the DF2SYS tape in the rear deck and press RESTART. The Interpreter is loaded automatically.

The Interpreter will respond to the commands discussed below. The form number is optional in most of these commands; if it is omitted, the current form will be used. An error may occur if no form is specified and none is currently in use.

<u>Only the first letter of the command word is needed</u>, thus, 'START 2' may also be entered as 'S 2'.

4.1 Data Tape Initiation

4.1.1 START New Data Tape

If the operator is preparing a <u>new data tape</u>, the:

START [n]

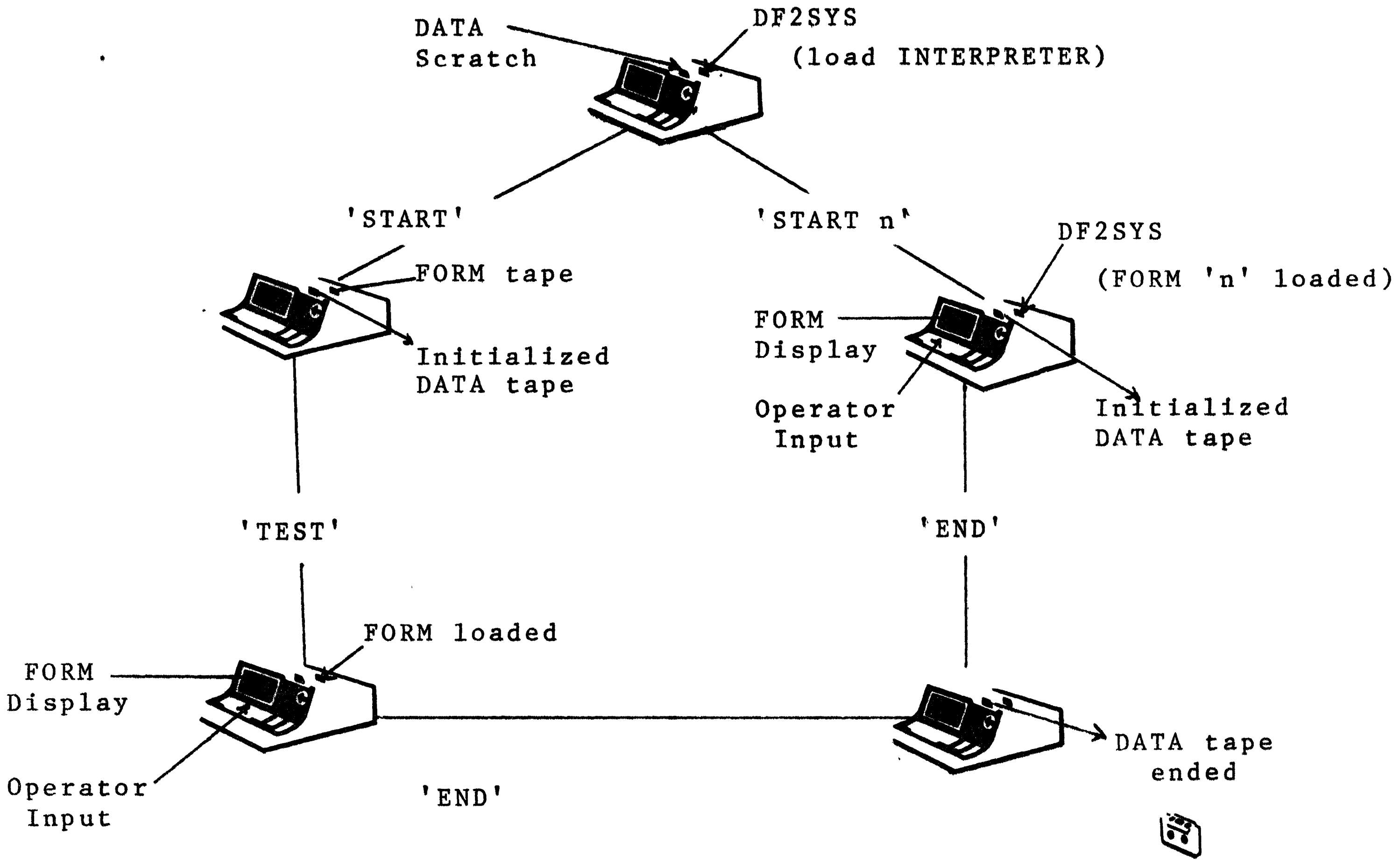
command must be excuted to initialize the tape. If a form is already loaded or is specified in the command, the form will be entered (i.e., the form is displayed with the cursor at the first non-constant data entry position available). Otherwise, control is returned to the monitor.

4.1.2 ADD to an Existing Data Tape

Similarly, data may be <u>added to the end</u> of an old data tape by entering the:

ADD [n]

command rewinds the data tape and. This command positions to the end of any data already on the tape. The form will be entered (if one is loaded or specified in the command) at the same time the data tape is being positioned. If there is no form present, control is returned to the monitor.



DATA ENTRY FLOW

A third way to open a data tape is to specify that it is an old tape to which changes must be made. This is indicated by executing either the MOD or the FIND command which are described later.

4.1.3 CONTINUE adding to an Existing Data Tape

If the data tape is very long and is already positioned at the end or in the midst of the data, the:

```
CONTINUE
```

command may be used. This command backspaces the tape once, to insure that it is in front of the end of file marker and then reads forward to the end of the data.

It is primarily intended to enable resumption of data entry mode after error corrections have been performed. 4.2 Loading a Form

4.2.1 Testing NEW Forms

A single form, not yet cataloged on the Dataform System tape, may be loaded using the:

```
NEW
```

command. The DF2SYS tape should be removed and replaced with the form tape _before_ typing ´NEW´.

4.2.2 LOADing Catalogued Forms

To initially _load a form_, from the system form library, or to replace the form currently in memory, the:

```
LOAD n
```

command may be used. Form ´n´ will be brought into memory and, if the data tape has been initiated, the form is entered; otherwise, the message ´TAPE CLOSED´ is displayed and control is returned to the monitor.

New forms may be loaded without disturbing the position of the data tape. Each data record contains the form number on which it was created so that subsequent modification or other processing can identify data generated on a particular form.

Note that the form is not reloaded if the number specified is the same as the current form.

Any time a form is loaded, a search for the form specified by the link is initiated (even if the linked form is not yet cataloged!). If the link is set to zero, no search is performed. Other tape operations, such as writing data, reading data, or positioning the data tape, cannot be performed until the linked file is found or all files on the system tape have been searched completely. That is, data may be entered on the form but may not be written to tape until the next form is found. If the form is not on the tape, the message 'BAD FORM' will appear the next time a tape operation (e.g. writing data) is performed.

Linked forms (and LOAD statements in field programs) are ignored if the form in memory was loaded using the 'TEST' command.

4.3 Entering DATA ENTRY Mode

To switch to data entry mode initially or to return to data entry mode from the monitor, the operator may type:

```
DATA
```

If no form is in memory or if the data tape is not open, an error message will be displayed and control will return to the monitor. Data currently in memory will not be disturbed and will be displayed whenever the form is re-entered.

4.4 Revising Existing Data Tape

4.4.1 MODIFY

Any data record on a finished data tape can be accessed for review or to be changed. The MODIFY command:

```
MOD [n]
```

(where n is an optional form number) enables the operator to manually access any data record created by a specified form and to then either bypass or change that record on the data tape. When the 'MODIFY' command is typed, the operator is asked to place a data tape in the front deck. The tape is searched for the first data record created by the current form. Once a record has been found, the data tape is in an 'open' mode and may be searched in a forward direction by pressing the read next record function key, or, from the monitor mode, by typing another 'MODIFY' command. To access records already passed over, use the rewind function key to rewind the data tape (like the initial MOD command).

During modification, a new form may be loaded (without disturbing the position of the data tape) and that form will subsequently be used for finding data records. Once a record has been found by the MOD command, the contents of all fields will be displayed in the form. Recorded data supercedes constants, thus, the actual data from the tape will be displayed; however, <u>constants will be set into the data record</u> when the field is entered (as they are for new records).

Data may be changed at this time by retyping the field. Press ENTER in the first column of a field to leave the data unchanged. The edit criteria and field programs associated with the form are still in effect.

4.4.2 Rewriting Existing Records

Data records are rewritten by the use of the write function key. If the record was fetched using the MODIFY comamnd, the next data record will automatically be read and displayed. If, however, the record was fetched by the FIND command, control is returned to the monitor.

Each data record contains a rewrite number. <u>When the rewrite number reaches 4, no further modification of that record is permitted</u>. The COPY command of the Configurator will reset rewrite counts so that data may be further modified. (The number of rewrites is limited to prevent loss of data due to tape deck positioning errors.)

If no field needs to be changed, the next record can be fetched by pressing the read next record function key; note that any modifications made will be destroyed by the read function. The write function <u>must</u> be used to cause updating of the record (unless the write is executed by the field program, in which case the field assigned the program must be entered).

4.4.3 FINDing a Specified Record

If unique data in the record to be corrected is known, the:

```
FIND [n]
```

command, where ´n´ is an optional form number, may be used. This command loads the specified form (if different from the current form) and displays the form so the operator may type characters into any fields to use as a key in searching the tape. No edit criteria or field programs are applied when setting up the match data. Thus, right justification and zero fill will not be performed.

When the data to be matched has been entered, the operator presses the read record function key to start the search. The interpreter will search the data tape forward looking for the record generated by the specified form and containing the specified data.

Once the matching data has been found, operation proceeds as in the MOD command. Note that this search may also be aborted by pressing the KEYBOARD and DISPLAY keys which returns control to the monitor.

If a match is not found, the message:

```
END OF DATA
```

appears and control is returned to the monitor.

4.4.4 Aborting Tape Searches

The tape search may be aborted by depressing both the KEYBOARD and DISPLAY keys. The operator may want to stop a search if, for example, the wrong data tape is in place, the wrong form is specified, or the wrong match data is given for a FIND. Control will be returned to the monitor.

4.5 Positioning the Data Tape

The data tape may not be positioned as long as it is in the ADD/START mode of data entry. The Read and Rewind function keys will be rejected.

4.5.1 Backspace Record Function

If, in the ADD/START mode, the Backspace Record function key is pressed, an end of file marker is written on the tape and the user is automatically switched to MODIFY mode before the tape is backspaced.

In the MODIFY mode, the Backspace Record function key causes the tape to backspace twice and read forward once under form number control, that is, if the record being read was not created by the current form, subsequent records will be read until a form number match is found.

The backspace record function may also be performed by typing the command:

```
BACKSPACE
```

This command executes exactly as does the function key.

4.5.2 Rewinding the Data Tape

There are two ways to rewind a tape during data modification. From the monitor, the operator may type the command 'REWIND'. This causes the data tape in the front deck to be rewound and positioned to the first data record created by the form currently loaded.

If, while viewing records during modification, the operator wants to rewind the tape in order to view records already passed, the rewind function key may be used.

4.6 ENDing the Data Tape

The operator cannot switch from the START or ADD mode to the MOD or FIND mode without writing end of file markers on the data tape. Return to the monitor via the System Control function key and use the:

```
END
```

command to 'close' the tape. If the operator is in the modification mode, the command will be rejected. The END command does not rewind the tape, nor does it clear any totals being accumulated by the current field program.

4.7 Data Entry Action

In the START, ADD, CONTINUE, DATA, MODIFY and FIND modes, the cursor will be positioned at the first free position of the first field. If, as in the modify mode, data is present, the current contents of the data record will be displayed.

Data set by a CONSTANT command at form generation is displayed and the cursor is placed at the first non-constant position on the form. However, data set by the SEMI-CONSTANT command when the form was generated will be displayed and the cursor will be placed in the the first position of the field (over the semi-constant).

If partial constants are set at the right hand end of the field, data must be typed up to the constants; otherwise, the constant data may be omitted in the output record.

During data entry, a CLICK sound is made for each accepted character. If a character fails to pass the edit criteria for the field (alpha, numeric or mixed) a BEEP is sounded and the cursor does not advance.

When typing data, pressing the ENTER key (or in fill-controlled fields, typing the last character) will cause the field to be further edited (right justified, zero filled, etc.) and, if no errors are found, the cursor will move to the next field. After the last field of a form is entered, the cursor will be placed back at the beginning of the first field awaiting a write function or other commands from the operator.

The ENTER key is used as a forward tab key and the Backspace Field function key permits backward tabbing. Forward tabbing past required fields is not permitted. Note that alpha/numeric editing occurs as the field is being keyed in. When the field is complete, further editing is performed on numeric and right justified fields to insure compliance with format restrictions (e.g., minus sign must be to the left of the field). User field programs are not executed until all other editing has been performed successfully.

4.8 Data Erase Function Key

The erase function clears the data area (without writing it to tape) and redisplays the cleared form. No indication is given to field programs that the erase function has been executed

4.9 Field Duplication Function Key

Once a form has been completed, the data is transferred to a second buffer from which it is written to tape. This secondary buffer is available to the operator for field duplication by means of the field duplication function key. If no previous record has been written, or if the preceding record was created by a different form, the results of pressing the duplication function key are not determined.

4.10 Operator Errors During Data Entry

When errors are detected in a field, instead of moving to the next field the cursor will be placed at the beginning of the field just entered and a beep will be sounded. The illegal data is not set in the data area in memory, but will still appear on the screen. If the operator decides to tab past the field, the last accepted data (blank if none has been entered) will be displayed.

4.11 Write Function Key

A data record will be written to tape whenever the Write Function key is pressed. The data record is written even if no data or only incomplete data has been entered. If, however, required fields have not been completed, the machine BEEPS and the cursor is placed at the first unfilled required field. The data is not written to tape. If an incomplete data record is written, it will contain zeros in all fields defined as zero filled (right justified, zero filled and left justified, zero filled) and spaces (or constants, if any) in all other unfilled fields.

After the current data has been written to tape, all data fields will be cleared to null values (or to the form constants or semi-constants if any) and the form will be reentered at the beginning. If, however, the auto-link flag is set (see AUTO LINKING in Section 2), when the write function is executed, the data is written out and the linked form is automatically loaded and displayed.

4.12 Loading Linked Forms

The next form (specified by the linkage information in the current form) will be brought into memory when the Load Form function key is pressed. The operator must record the current data record (using the Write function key) prior to loading the next form, since pressing the Load Form function key does <u>not</u> write and clears any data in memory.

Control can be returned to the monitor from the data entry mode by use of the System Control function key.

4.13 Returning Control to the Monitor

Whenever, while entering data into a form, it becomes necessary to type one of the Interpreter 'commands' listed in Appendix C, the operator must press the System Control function key to return control to the monitor. Only then may the needed command be typed.

5.0 RECOVERY PROCEDURES

5.1 DATAFORM SYSTEM TAPE RECOVERY

The primary rule of system security is keep BACKUP tapes; i.e., copy DF2FGS, DF2SYS and DF2PGS using the COPY utility program. Once forms have been cataloged, copy the Dataform system tapes using the DUP-ALL command. Master form and program tapes should be carefully labeled and stored.

If parity errors develop in the system program section of the tape, the RIN command can be used to copy the forms onto another Dataform system.

If errors develop in the forms themselves it may be possible to REPLACE or DELETE or CHOP the erroneous areas.

Parity errors in the forms catalog will cause the message:

FORMS CATALOG UNLOADABLE, DUMMY CATALOG GENERATED

to appear. If CAT is typed, the dummy catalog that assumes all forms, 1-123, will be displayed. If the forms on the tape are consecutive one may CHOP the last actual form + 1. This will cause the catalog to be rewritten, hopefully eliminating the previous error. One may also attempt to salvage the forms by OUT'ing them one at a time.

5.2 DATA TAPES

During data entry, each record is written, re-read and compared to the original data. If a failure occurs, the operator is informed that the tape has been ended prior to the record containing the error. A new data tape must be initiated (using START) and the last record (which may be retrieved using the field duplication function key) and the one in progress when the failure occured must be re-keyed and re-written.

Three types of problems can arise with data tapes:

parity errors
missing end of file markers
or rewrite limit reached.

Since read-after-write technique is used to write data records initially, undetected parity errors should be rare. These will most likely occur during error correction modification.

The COPY command can be used in any of the above cases to correct the tape. The rewrite count is automatically reset by the COPY command. Parity errors may be omitted or copied (in the hopes of later correcting the record by modification). If a file marker is missing, a parity error will usually be encountered, in which case the END option of the copy command may be used.

Another technique for adding a lost or omitted end of file marker is to use the Interpreter:

Find the last record using the MOD command (or FIND if the actual data is known).

Remove the data tape.

Place a scratch tape in the front deck, and type START. Once the starting file marker has been written, the scratch tape is replaced by the actual data tape.

Type END; this will write an end of file after the last record.

5.3 FORM TAPES

If, during the intial generation process, an unrecoverable error occurs at the point of writing the form, the Generator, DF2FGS, may be reloaded _immediately_ and the RECOVER command executed. This command causes the form still in memory to be accessed. One of the form generation passes, e.g., TYPE, may be executed to insure that the form is still intact. Then try OUT'ing the form again, using a new cassette in the front deck.

If one of the system files could not be loaded, use another copy of DF2FGS to recover the form.

Appendix A
Type codes

TYPED CODE	MEANING
A	Alpha (A - Z and space)
D	Digit (0 - 9)
N	Numeric (0 - 9, period and minus)
M	Mixed alpha and numeric
L	Numeric,Left Justified/Zero filled
R	Numeric,Right Justified/Zero filled
B	Numeric,Right Justified/Blank filled
CANCEL	Clears edit criteria

Right justified fields are filled with leading zeros (R) or blanks (B). During data entry, the field is justified and re-displayed after the ENTER key is pressed. Numeric fields are limited to 12 places of significance to the left and 4 places to the right of the decimal point.

Require Codes

TYPED CODE	MEANING
R	Required (at least 1 character must be keyed)
F	Fill Controlled (all characters must be keyed) (ENTER key allowed only to bypass field)
B	Both Fill Control and Require
P	Program Reserve (no keyin)
S	Required and Program Reserve (field is checked prior to write)

Appendix B
Image Generation Function Keys

There is a set of function keys available in the image generation mode only. When the DISPLAY key is pressed, certain characters become function keys. These function keys can all be found on the number pad. The following functions are available:

7 - character insert
8 - up cursor
9 - erase to end of frame
4 - left cursor
5 - character duplicate
6 - right cursor
1 - word remove
2 - down cursor
3 - form expand (downward)
0 - character remove
. - erase to end of line

The BACKSPACE key and DISPLAY/4 have the same function of non-destructive left cursor movement. Backspacing from column 1 back to column 80 is permitted.

The CANCEL key erases the entire line the cursor is on and places the cursor at the beginning of the line.

The KEYBOARD key acts as a REPEAT key for all characters and for most function keys.

The DISPLAY/CANCEL function causes an edit table to be generated with the field position and length set and all edit conditions set to default values.

NUMBER PAD OVERLAY

Char Insert	UP	Erase Frame
LEFT	Dup Char	RIGHT
Word Remove	DOWN	Form Expand
Character Remove		Erase Line

Cut this overlay out and use for reference during form generation.

Appendix C
Data Entry Interpreter Function Keys

Mode	Key Typed	Function
All Data Entry	DISPLAY/4	return to monitor
	DISPLAY/.	write data record or rewrite it
	ENTER	forward tab
	DISPLAY/3	backward tab
	DISPLAY/6	erase data area
	DISPLAY/1	load next form
Modify and Find	DISPLAY/7	rewind data tape
	DISPLAY/8	backspace record
	DISPLAY/9	read record

Appendix D
Error Messages

SYSTEM ERRORS

FILE MISSING

Some form, present in the catalog, is missing on the system tape, or the file marker necessary for positioning the input tape is missing, or a form is short (i.e. it doesn´t contain the necessary 6 blocks).

BAD NUMBER

The form number may have been omitted, out of range (1-123) or non-numeric. Or, the form specified is not in the catalog. Note that if the form number is omitted in a command which optionally accepts form numbers (e.g. START [n]) the command line cannot end with a space.

In the Interpreter, this message may mean that the next form specified (in the current form´s link) is not in the catalog, or that your command assumes that there is a form in memory (e.g. ENTER) and none is loaded.

PARITY ERROR ON DECK X

Indicates a parity error was detected - where X is the deck number (1 = system tape, 2 = front deck). Before this message is displayed, four attempts are made to read the record.

INTERNAL ERROR n ON DECK m

This message indicates a tape or tape deck failure. The n is replaced by a letter indicating the error condition:

D - parity error
E - end of tape
F - end of file
G - unfindable file
Z - write failure

Generally these errors occur only if something is severly wrong with the cassette. Error Z may occur if the write protect tab has been punched on the cassette or if the tape is improperly inserted in the deck. If error Z occurs often a hardware failure should be suspected.

The letter ´m´ in the message is replaced by the number of the tape deck on which the error occured (deck 1 is the rear deck, deck 2 is the front).

CONFIGURATOR ERRORS

END OF FILE MISSING
: End of tape reached during COPY - an end of file marker is automatically written.

AUTO NOT SET
: Is given in response to a MANUAL command if the auto-load entry is not set.

FORM CATALOG UNLOADABLE, DUMMY CATALOG GENERATED
: File 1, the forms catalog is in error, a dummy (full) catalog has been substituted. Steps should be taken to recover the system (see section 5).

NUMBER IN USE
: The form number specified for an IN command was already assigned.

GENERATOR ERRORS

BAD FORM
: The form just written is unloadable due to parity errors or missing blocks.

MORE THAN 126 FIELDS
: During image generation more than 126 data fields were defined. The form must be revised before it may be written out.

MORE THAN 245 DATA
: During image gneeration more than 245 data characters were defined. The form must be revised before it may be written out.

XXX DATA
YYY BYTES LEFT
: The messages appear immediately after the image generation phase of form generation. They is for information only.

YYY BYTES OVER
: If this message appears after image generation, the form image, data area and edit table have combined to overflow the user space. Something must be reduced.

INTERPRETER ERRORS

Continuous Beeping during data entry
: An illegal constant has been defined at form

generation time. The constant must be reset to conform with the edit criteria before proceding.

Continuous Clicking during data entry
An all constant form with no keyin field has been loaded. The form must be corrected before data entry may proceed.

TAPE CLOSED
No START, ADD, MOD or FIND command has been executed.

END OF TAPE
End of tape was encountered during data entry or an unrecoverable tape error occurred during writing. If the error occurs due to end of tape, the data tape is automatically backspaced twice and an end of file marker is written. If it is a write-parity error, the end of file is written where the record would have been. This means the last two records (since the operator is keying one in) are lost. Totals being accumulated by field programs may no longer be valid.

REWRITE LIMIT REACHED TO VIEW PRESS ENTER
During modification, the record in memory has been rewritten 4 times and cannot be written again; however, it can be interrogated. To reset the rewrite counter to zero, use the COPY command of the Configurator.

BAD DATA
There is a parity error in the data.

BAD FORM
There is a parity error in the form or a block is missing.

END OF DATA
End of file has been reached on the data tape.

TAPE OPEN
An ´open´ type operation was attempted before ENDing the current data tape.

Appendix E
COMMANDS

<u>CONFIGURATOR:</u>	
CATALOG	display the forms in the catalog
CHOP	delete specified form and all subsequent forms
COPY	copy a data tape and reset rewrite counters to zero.
DELETE	delete the specified form
DPRINT	print data tape
DUP	duplicate the main system with a blank catalog
DUP ALL	duplicate the entire system including forms
FPRINT	print form
IN	input a form assigning the specified form number
INTERPRETER	run the Dataform Interpreter
LGO	write faster loading Interpreter
OUT	output the specified form
REPLACE	replace the specified form
RIN	replace the catalog and forms with the catalog and forms of another Dataform system.
<u>GENERATOR:</u>	
CONSTANT	set constants into the form
LINK	define next form linkage
NEW	clear the work area for a new form
OLD	load old form from front deck
OUT	write the current form record to tape
PROGRAM	assign program letters to fields
REQUIRE	set required fields
REVISE	revise the current form
SEMI-CONSTANT	set semi-constant data into the form
TYPE	set edit keys for the current form
<u>INTERPRETER:</u>	
ADD	add to the end of a data tape
BACKSPACE	backspace a record on data tape
CONTINUE	resume data entry after modification
DATA	switch to data entry mode
END	write an end of file on the data tape
FIND	search for matching data record
LOAD	load the specified form
MODIFY	modify data records
NEW	load un-cataloged form
REWIND	rewind data tape
START	initialize a data tape

Appendix F
Sample Form Generation

```
                          EMPLOYEE PAYROLL RECORD

Name |^^^^^^^^^^^^^^^^^^^^^^^^^^^^^    Title Code |^^  Dept |^
     Dependents |^  State Code |_      Social Security |^^|^^^^^
     Exempt/Nonexempt (0/1)    |      Workman's Compensation (0 to 9) |
     Married/Single (0/1)      |      Male/Female (0/1)  |

Hourly Rate $|^^^^^  Amount Last Increase $|^^^^^^  Date Last Increase |^^^^^
Date Hired   |^^^^^        Date Terminated |^^^^^       Date of Birth  |^^^^^
State Tax    |^^^^^        Disability Tax  |^^^^^       City Tax       |^^^^^
Insurance    |^^^^^        Auto Insurance  |^^^^^       Life Insurance |^^^^^
     Advance |^^^^^        FICA Status (exempt=0, nonexempt=1) |    Page 2? <
```

Sample Form - During Form Generation
NEW or REVISE command

Form text, data and keyin field definitions are set. If no constants or semi-constants are added, this is the way the form text will look during data entry except that the carets will be replaced by spaces.

```
                         EMPLOYEE PAYROLL RECORD

Name A                                  Title Code D    Dept D
     Dependents D   State Code D_       Social Security D  D
     Exempt/Nonexempt (0/1)    D        Workman's Compensation (0 to 9) D
     Married/Single (0/1)      D        Male/Female (0/1)  D

Hourly Rate $R     Amount Last Increase $R        Date Last Increase  D
Date Hired   D           Date Terminated D           Date of Birth    D
State Tax    R           Disability Tax  R           City Tax         R
Insurance    R           Auto Insurance  R           Life Insurance   R
     Advance R           FICA Status (exempt=0, nonexempt=1) D   Page 2? A
```

Sample Form - During TYPE pass

Edit types are set. These will not be displayed during data entry.

```
                       EMPLOYEE PAYROLL RECORD

Name ¦                                      Title Code ¦    Dept ¦
     Dependents ¦   State Code 42           Social Security ¦  ¦
     Exempt/Nonexempt (0/1)     1           Workman's Compensation (0 to 9) ¦
     Married/Single (0/1)       0           Male/Female (0/1)  1

Hourly Rate $¦       Amount Last Increase $¦         Date Last Increase ¦
Date Hired   ¦             Date Terminated ¦             Date of Birth  ¦
State Tax    ¦             Disability Tax  ¦             City Tax       ¦
Insurance    ¦             Auto Insurance  ¦             Life Insurance ¦
     Advance ¦             FICA Status (exempt=0, nonexempt=1) 1
```

Sample Form - During SEMI-CONSTANT pass

Several fields are preset to commonly entered values. These may be typed over by the operator. The CONSTANT pass looks the same, however, constants may not be typed over during data entry.

```
                         EMPLOYEE PAYROLL RECORD

Name R                                   Title Code B    Dept B
     Dependents B   State Code F         Social Security R  R
     Exempt/Nonexempt (0/1)    F         Workman's Compensation (0 to 9) F
     Married/Single (0/1)      B         Male/Female (0/1)  B

Hourly Rate $R       Amount Last Increase $¦         Date Last Increase F
Date Hired   B             Date Terminated F             Date of Birth  F
State Tax    R             Disability Tax  ¦             City Tax       R
Insurance    ¦             Auto Insurance  ¦             Life Insurance ¦
     Advance ¦             FICA Status (exempt=0, nonexempt=1) B   Page 2? <
```

Sample Form - During REQUIRED pass

This pass indicates if fields are required, fill-controlled, program reserved, etc.

```
                         EMPLOYEE PAYROLL RECORD

Name ¦                                      Title Code ¦    Dept ¦
     Dependents ¦   State Code ¦_           Social Security ¦   ¦
     Exempt/Nonexempt (0/1)    A            Workman's Compensation (0 to 9) B
     Married/Single (0/1)      A            Male/Female (0/1)  A

Hourly Rate $¦         Amount Last Increase $¦         Date Last Increase D
Date Hired   D               Date Terminated D             Date of Birth  D
State Tax    ¦               Disability Tax  ¦             City Tax       ¦
Insurance    ¦               Auto Insurance  ¦             Life Insurance ¦
     Advance ¦               FICA Status (exempt=0, nonexempt=1) A   Page 2? X
```

Sample Form - During PROGRAM pass

Program 'A' checks range 0-1, 'B' checks 0-9, 'D' checks for valid dates and 'X' checks for a 'Y' or 'N' to determine if another form should be loaded.

SECTION II

DATAFORM LANGUAGE

TABLE OF CONTENTS
SECTION II

1.0 Elements of the Language
- 1.1 Labels 1
- 1.2 Pre-defined labels 1
- 1.3 Field program names 1
- 1.4 Blanks 2
- 1.5 Comments 2

2.0 Specification Statements
- 2.1 DATA -- Accessing the output buffer 3
- 2.2 WORK -- Intra-form work areas 3
- 2.3 COMMON -- Inter-form work areas 6
- 2.4 EQU -- External definitions 6
- 2.5 REDEFINE -- Redefinition of work areas 7
- 2.6 FIELD -- References to specific field numbers 8

3.0 Executable statements
- 3.1 Transfers of information 9
  - 3.1.1 ALIGN 9
  - 3.1.2 CONVERT 10
  - 3.1.3 LOOKUP 10
  - 3.1.4 MOVE 11
  - 3.1.5 SET 11
- 3.2 Arithmetic operations 12
- 3.3 Comparisons 13
- 3.4 Output control 14
  - 3.4.1 WRITE 14
  - 3.4.2 MESSAGE 14
  - 3.4.3 SHOW 15
  - 3.4.4 BEEP 15
  - 3.4.5 CHAIN 16
  - 3.4.6 FORMSHOW 16
- 3.5 Transfers of control 16
  - 3.5.1 GOTO 16
  - 3.5.2 CALL and RETURN 17
  - 3.5.3 CHANGE and RESET 17
  - 3.5.4 Pre-defined Labels 17

4.0 Pre-defined labels
- 4.1 AGAIN 19
- 4.2 END 19
- 4.3 INPUT 19
- 4.4 NEXT 19
- 4.5 NULL 19
- 4.6 OUTPUT 20
- 4.7 RETRY 20
- 4.8 STORE 20

5.0 Program Generation
    5.1 Editing a source program 21
    5.2 The compilation process 21
    5.3 The compilation listing 22
    5.4 The program tape 23
    5.5 Form generation 23
    5.6 Testing 24

6.0 Program Execution
    6.1 Post process execution 25
    6.2 Operator tabbing 25
    6.3 Pre-process execution 25
    6.4 Program reserved fields 25
    6.5 Form constants 25

APPENDIX A: Sample programs

APPENDIX B: Error messages

APPENDIX C: Commands requiring the extended interpreter

APPENDIX D: Code reduction techniques

APPENDIX E: Instruction list

1.0 Elements of the language

The DATAFORM 2 language consists of two kinds of statements: executable statements (i.e., those that do actual data manipulation), and specification statements (i.e., those that describe the different kinds of data available).

1.1 Labels

Any statement which is referenced elsewhere in the program will have a label. A label:

1) begins in column 1.
2) consists solely of alphanumeric characters.
3) may be any number of characters in length, although all characters after the first eight are ignored.

1.2 Pre-defined labels

Seven labels are pre-defined to the compiler. These labels are listed here, and discussed more fully below in section 4.0.

1. AGAIN	5. NULL
2. END	6. OUTPUT
3. INPUT	7. STORE
4. NEXT	8. RETRY

If any of these labels is re-defined in the program, the re-definition will be flagged as a duplicate label error and the re-definition will be ignored. The maximum number of labels allowed by the compiler, including pre-defined labels, is 100. Any labels defined after this maximum is reached are ignored. Examples of labels:

```
A
2765
FIELD17
LABELSTATEMENT       (truncated to LABELSTA)
```

1.3 Field program names

A field program name is the address at which the interpreter begins execution of a program. A field program name is defined as a label immediately succeeded (no intervening blanks) by an asterisk (*). Only the first character of a field program name is passed to the interpreter; therefore, program names should be only one character in length. In addition, the interpreter deals only with alphabetic program names; therefore, the one character should be alphabetic. The compiler does not check for duplicate program names for the same character; if there are duplicates, it passes both to the interpreter. Since program names must be alphabetic and only one character in length, the maximum number of program names that the interpreter deals with is 26. Examples of program names:

```
E*
Z*
```

1.4 Blanks

One or more blanks are treated as a field separator. A label must be separated from an operand (with the exception of the program name asterisk described above) by at least one blank. Blanks are ignored except as field separators.

1.5 Comments

Comments are of two types -- entire line comments, or partial line comments. Entire line comments begin with a period in column 1. Any line which has a period in column 1 is ignored entirely by the compiler. Partial line comments may begin on a line whenever the syntax for that line is complete. Partial line comments should also begin with a period, and are allowed on all statements except where noted in the discussion of the individual statements below.

A line that begins with a plus sign (+) is a comment with the special property of forcing that line to begin at the top of a new page.

2.0 Types of specification statements

The six types of specification statements are: DATA statements; WORK statements; COMMON statements; EQU statements; REDEFINE statements; and FIELD statements.

2.1 DATA -- Accessing the output buffer

DATA statements refer to specific <u>columns</u> of the output data record (the form). The general format of the DATA statement is:

```
        <label>     DATA n,m
```

where ´n´ and ´m´ are decimal numbers in the range 1-245. The number ´n´ refers to an initial column of the output data record, and the number ´m´ refers to a terminal column of the output data record. The columns defined by the DATA statement need not correspond to specific fields of the form. Areas may be redefined. Therefore, columns defined by the DATA statement may be:

1) identical to fields on the form;
2) a sub-grouping of a large field into smaller fields;
3) a combination of smaller fields into a larger field;
4) an overlapping of fields on the form.

Syntax restrictions:

1) ´n´ and ´m´ must be greater than zero but less than 246.
2) ´m´ must be greater than or equal to ´n´.
3) the DATA statement must have a label.

Examples:

```
        NAME        DATA 1,29    multiple column field
        IDCODE      DATA 30,30   single column field
        AMOUNT      DATA 31,39
        DOLLARS     DATA 31,37   Sub-group of larger
        CENTS       DATA 38,39        field
```

2.2 WORK -- Intra-form work areas

The WORK statement is used to reserve space within a field program for both non pre-defined and pre-defined working storage (the latter consisting of ASCII or octal constants, or tables). To reserve non pre-defined working storage, the following format is used:

```
        <label>     WORK n
```

where ´n´ is a decimal number in the range 1-245 inclusive. The area called <label> would have an item length of ´n´.

Working storage is pre-defined with ASCII characters by inclosing the desired characters in double quotation marks. Examples:

```
        <label>     WORK "PRE-DEFINED CHARACTERS"
```

A special forcing character, ´#´, is used to ´force´ the character immediately following it to be included in the string; by using this character, the double quotation mark and the forcing character may appear in the character string:

```
        <label>     WORK "FORCED DOUBLE QUOTE: #" AND"
        <label>     WORK "FORCED FORCING: ##"
```

Each WORK statement that is pre-defined generates a code segment. Every pre-defined working storage segment is by default terminated with an additional, special end-of-table character, an octal zero. This character is included in computing the over-all length of the working storage segment, but is not included in computing the item length. It is possible to suppress the special end-of-table character in a pre-defined working storage statement by following the terminal double quotation mark with a semicolon. Examples:

```
        WORK1       WORK "DATA"
        WORK2       WORK "DATA" ;
```

The first example will generate the following five bytes: 0104,0101,0124,0101,000. The second will generate the following four bytes: 0104,0101,0124,0101. The individual item lengths are not affected by the semicolon; only the amount of code generated is affected.

Tables are pre-defined by inclusion between double qoutation marks as well. The item length of the table is determined by the length of each item in double quotation marks. In these four examples:

```
        LABEL1      WORK "1","2","3","4","5","6"
        LABEL2      WORK "12","34","56"
        LABEL3      WORK "123",456"
```

```
LABEL4      WORK "123456"
```

all of the working storage tables have the same table length (six characters plus one special end-of-table character for a total table length of seven), but the individual item lengths are respectively 1, 2, 3, and 6. The overall length is not used for table lookups or range checks; table length, however, is meaningful for the CONVERT and the table and range IF comparisons.

Pre-defined working storage areas may have an overall length greater than can be contained on one input code line; therefore working storage items may be continued on more than one line by using a colon, as in this example:

```
LABEL1      WORK "123456","789012":
                 "345678":
                 "901234"
```

Working storage may be predefined with octal values. This is done by presenting one or more octal constants, the first of which (and only the first) is prefixed by the alphabetic letter "O". Each octal constant generates only one byte of working storage. An octal constant may consist of a string of any number of octal digits; however, only the least significant eight bits of the string are placed in the generated byte. Octal constants may be grouped together using a comma to separate each constant, and may be continued from one line to another by terminating the first line with a colon. Each octal work area is succeded (by default) by a single binary zero, inserted automatically by the compiler, which is used as a table terminator. A semicolon following the last octal constant will suppress the default zero generation. The item length of an octal constant area is one. Octal constants and ASCII character strings may not be mixed in the same WORK statement; WORK statements are either octal or ASCII. Examples of octal WORK statements:

```
<label1>    WORK O15
<label2>    WORK O15,16,17,20
<label3>    WORK O15,16,17,20:
                  25,26,27,30:
                  35
```

Syntax restrictions:

1) the WORK statement must have a label.
2) in a table, all items must be of the same length.
3) line comments may appear on WORK statements if the comment is preceeded by a period.
4) the length of non pre-defined areas may not

exceed 245. The total length of a pre-defined table can be greater than 245, although the maximum item length is what can be packed on one line.

2.3 COMMON -- Inter-form work areas

COMMON statements are identical syntactically to WORK statements. Their main difference is one of function. The COMMON area is used for transferal of information between forms, or for the saving of information used in one form only, although multiple forms are loaded.

It is important for every program using information saved through COMMON to have the same relative locations of areas inside the COMMON block. The statement:

```
<label>    COMMON n
```

should be used to skip over ´n´ unused bytes inside the COMMON block, if those bytes are not referenced by the current form, and are referenced by another form.

Syntax restrictions:

1) COMMON statements need not have labels, unless the block of information is to be referenced in the program.
2) the maximum total length of the COMMON block is 100 bytes.
3) line comments may appear on COMMON statements if they are preceeded by a period.

2.4 EQU -- External definitions

The EQU statement is used to define absolute octal addresses. Following the EQU is a string of octal digits, denoting an absolute octal address. The initial character of the string need not be a zero, although a zero will serve as a reminder that the string is octal rather than decimal.

If the system has more than minimal (minimal = 8K) memory, this extra memory may contain previously assembled assembly (as distinct from Dataform) language programs, which may be referenced by using the EQU statement to define a label, and then transferring control to that label (see section 3.5 of this manual for transfer of control statements and Section III for assembly language interfacing). Examples:

```
8K          EQU 020000
12K         EQU 30000
```

2.5 REDEFINE -- Redefinition of buffer and work areas

The REDEFINE statement is used to associate a new label with an elsewhere defined label.

The general format of the REDEFINE is:

```
<label2>    REDEFINE label1,n,m
```

The number ´n-1´ is added to the previously defined address for label1 and becomes the initial address of label2. The item length of label1 is ignored, and the number ´m´ becomes the item length for label2.

For example, suppose a table is defined as follows:

```
TABLE1      WORK "123456789012"
```

The item length of TABLE1 is 12. Then consider:

```
TABLE2      REDEFINE TABLE1,1,6
TABLE3      REDEFINE TABLE1,1,4
TABLE4      REDEFINE TABLE1,1,3
TABLE5      REDEFINE TABLE1,3,2
TABLE6      REDEFINE TABLE1,7,1
```

The same memory locations are "re-grouped" under different labels, so that the effect is the same as:

```
TABLE2      WORK "123456","789012"
TABLE3      WORK "1234","5678","9012"
TABLE4      WORK "123","456","789","012"
TABLE5      WORK "34","56","78","90","12"
TABLE6      WORK "7","8","9","0","1","2"
```

The REDEFINE is not restricted to WORK statements; COMMON and DATA statements may also be REDEFINE´d.

Syntax restrictions:

1) ´n´ and ´m´ must be greater than zero but less than 245.
2) the REDEFINE must have a label.
3) partial line comments may appear after the ´m´.
4) The REDEFINE statement should immediately follow the label that is being redefined

(i.e., label1 in the general format of the REDEFINE above). The REDEFINE statement is not flagged in error if it appears elsewhere, but erroneous values may be generated if the REDEFINE succeeds the statements that reference the REDEFINE´d label.

2.6 FIELD -- References to specific field numbers

The FIELD statement assigns names to areas of the output record corresponding to on the displayed form. Two types of field references are provided -- absolute and relative. The absolute format is used to reference <u>specific</u> fields of the form. Its format is:

```
          <label>     FIELD n
```

where the ´n´ is a decimal number from 1-245.

The relative format is used to reference an offset (either positive or negative) of the <u>current</u> field. The relative format is:

```
          <label>     FIELD [sign]n
```

where [sign] is either a ´+´ or a ´-´; and ´n´ is a decimal number from 1-245. For example:

```
          FLD7        FIELD 7
          NXTFLD      FIELD +1
          LSTFLD      FIELD -1
```

Labels assigned via FIELD statements may be referenced in any type of arithmetic or conditional statement. For example:

```
                      ADD LSTFLD TO INPUT GIVING NXTFLD
```

3.0 EXECUTABLE STATEMENTS

Executable statements are all statements other than specification statements. They have to do with: 1) transfers of data; 2) arithmetic; 3) comparisons; 4) output; and 5) transfers of control.

In the subsequent discussion of generated code formats, all addresses are two bytes in length, and are stored as MSB, LSB. Relocatable addresses have the sign bit of the MSB set to a one. Absolute addresses are all addresses other than relocatable addresses, and are as they were defined.

3.1 Transfers of information

Data is moved from one location to another using one of six possible verbs: ALIGN, CONVERT, LOOKUP, EDIT, MOVE, or SET.

3.1.1 ALIGN

The ALIGN verb format is:

```
<LABEL>     ALIGN field1 TO field2
```

The ALIGN first checks both field1 and field2 for the presence of a decimal point. If none exists, it is assumed to be at the rightmost edge of the field. After determining the decimal point, field1 is moved to field2, with decimal points aligned. In field2, either truncation or zero-fill or both may occur. Examples:

FIELD1	FIELD2 (before)	FIELD2 (after)
10.1	0000.	0010.
10.1	00.00	10.10
10.1	0.000	0.100
1.234	0000.	0001.
1.234	00.00	01.23

NOTE: If field2 is in the data area, the decimal format may be initialized by setting (during form generation) semi-constant zeros with a decimal point in the appropriate position.

CODE: The ALIGN generates 7 bytes of code: op, field2 length, field2 address, field1 length, field1 address.

3.1.2 CONVERT   Extended

The CONVERT verb format is:

```
        <label>     CONVERT field1 BY table1 AND table2
                              GIVING field2
```

The CONVERT verb will try to find field1 in table1. The length of field1 is used for the search. The corresponding entry in table2 is moved to field2. Examples:

TABLE1	TABLE2
01	JAN
02	FEB
03	MAR
04	APR
etc	etc

After a convert,

FIELD1	FIELD2
04	APR
07	JUL
01	JAN

The item length of table2 is used to determine the position of the corresponding element and the length of the move from table2 to field2 (the item length of field2 is also checked); therefore, each separate item in table2 should be enclosed in double quotation marks.

If the item is not found in table1, no movement of data takes place.

CODE: The CONVERT generates 12 bytes of code: op, table1 address, field1 length, field1 address, field2 length, field2 address, table2 length, table2 address.

3.1.3 LOOKUP   Extended

The LOOKUP verb format is:

```
        <label>     LOOKUP field1 IN table1 GIVING field2
```

The LOOKUP verb will use field1 as an index into TABLE1. The item thus selected will be moved to field2. If the index value is greater than the length of the table, the value moved into field2 is indeterminate. The example of section 3.1.2 above could be coded as:

```
<label>     LOOKUP field1 IN table2 GIVING field2
```

The LOOKUP verb should be used when there are no 'gaps' in the table from which the data movement takes place. The CONVERT verb should be used when the table has gaps, or is randomly ordered. The LOOKUP uses field1 as an item by item index into the table, and hence will always find a match; the CONVERT verb searches through the first table until it finds a match.

CODE: The LOOKUP generates 10 bytes of code: op, table1 length, table1 address, field1 length, field1 address, field2 length, field2 address.

3.1.4 MOVE

The move verb format is:

```
<label>     MOVE field1 TO field2
```

Field1 is moved, left justified, to field2. If the length of field1 is less than the length of field2, field1's length is used in the move. Subsequent characters in field2 are not changed; their values are as they were before the MOVE. If the length of field2 is less than the length of field1, field2's length is used.

CODE: The MOVE generates 7 bytes of code: op, field2 length, field2 address, field1 length, field1 address.

3.1.5 SET

The SET verb format is:

```
<label>     SET field1 to field2
```

The first character of field2 is spread throughout field1 -- as for zeroing out a total, or blank filling a message. The example:

```
ASTERISK    WORK "*"
TOTAL       WORK "00000000"
<label>     SET TOTAL TO ASTERISK
```

would set the entire 8 character total field to asterisks. This should not be used to zero a field containing a decimal point which is to be used as a destination for ALIGN or any arithmetic statements since that decimal will be overstored.

CODE: The SET generates 6 bytes of code: op, field2 address, field1 length, field1 address.

3.2 Arithmetic operations

MULTIPLY - Extended
DIVIDE - Extended

The standard arithmetic functions of add, subtract, multiply and divide are provided. These statements must be in the following formats (specifically, the connectives between label1 and label2 must not vary):

```
<label>     ADD label1 TO label2
<label>     SUBTRACT label1 FROM label2
            (SUBTRACT may be abbreviated SUB)
<label>     MULTIPLY label1 BY label2
            (MULTIPLY may be abbreviated MUL or
                  MPY)
<label>     DIVIDE label1 INTO label2
            (DIVIDE may be abbreviated DIV)
```

Alternatively, any of the above four may be modified by appending the phrase <GIVING label3> to them. The result of this is that the contents of the first two labels are not affected, but their sum (difference, product, qoutient) appears at the third label rather than the second. Examples:

```
<label>     ADD label1 TO label2 GIVING label3
<label>     MPY label1 BY label2 GIVING label3
```

This format causes an 'ALIGN label2 TO label3' to be generated prior to the arithmetic statement.

NOTE: Significance may be lost at this point (before computation) if label3 has fewer places of significance than label2.

The result of any arithmetic will be aligned to the decimal point in the result field. Truncation is performed at both ends of the field and leading <u>zeros</u> are supplied if necessary; thus, in a field defined as right justified and blank filled, performing an 'ADD NULL TO field' will replace the leading blanks by zeros.

CODE: If the arithmetic statements have a GIVING appended, they will generate a 7 byte ALIGN of label2 to label3 before generating the normal 7 bytes of code for the arithmetic operation: op, label2 length, label2 address, label1 length, label1 address.

3.3 Comparisons

CK10 - Extended
CK11 - Extended

The general format of the comparison is:

```
        <label>   IF field1 [RELATION field2 THEN label1
```

If the relation is true, control is transfered to label1. Three types of relations may be defined:

1) ASCII comparisons (EQ, EQU, EQUAL, GE, GEQ, GREATER, GT, GTR, LE, LEQ, LESS, LESSTHAN, LT, NE, NEQ, NOTEQUAL are all acceptable). The character values in field1 are compared, from left to right, to the characters in field2 (<u>using the length of field1 to terminate the compare</u>). Differing lengths do not cause unequal compares; however, if field1 is longer than field2, the results are indeterminate.

2) table lookup (INR, INRANGE, INT, INTABLE, NIR, NOTINRANGE, NIT, NOTINTABLE). Field1 is "looked-up" in the table defined at field2. The length of field1 is used.

3) check digit verification. Field1 is tested for correctness of check digit with either a mod 10 (CK10) or a mod 11 (CK11) check performed, using the contents of field2 as a weighting factor. Field1 should contain the check digit in the least significant position. Field2 is assumed to be one character shorter than field1.

Examples:

```
FIELD1       FIELD 1
ACCOUNTNO    DATA 21,27
MONTH        DATA 1,2
DAY          DATA 3,4
DAYTABLE     WORK "01","31"
MONTHTABLE   WORK "01","02"
ZERO         WORK "000000"
WEIGHT1      WORK "212121"
.
. Check field1 for strictly positive
.
A*           IF FIELD1 GREATER ZERO THEN STORE
             AGAIN
.
. Check for null input
.
B*           IF NULL EQ INPUT THEN AGAIN
```

```
.
. Check for negative.
.
C*             IF FIELD1 LT ZERO THEN STORE
               AGAIN
.
. Check range using table
.
.
D*             IF DAY NOTINRANGE DAYTABLE THEN AGAIN
               IF MONTH NIR MONTHTABLE THEN AGAIN
               STORE
.
. Perform Mod10 check digit validation
.
E*             IF ACCOUNTNO CK10 WEIGHT1 THEN STORE
               AGAIN
```

CODE: The IF generates 8 bytes of code: op, field2 address, field1 length, field1 address, label1 address.

3.4 Output control

Three output statements are provided: WRITE, MESSAGE and SHOW. (See also END and CLOSE in Section 4.) The BEEP statement provides an audible tone, and the LOAD statement is used to load another form (in addition to the auto-load and linking-load features of the interpreter).

3.4.1 WRITE

The WRITE statement will write the data area to the front cassette deck. Control is returned to the next statement in the user's program. (See also END in section 4.) The data area in memory is not cleared, and may be passed to another form for further computation or may be used for auto-duping selected data. Its format is:

```
<label>     WRITE
```

CODE: The WRITE generates only a 1 byte op code.

3.4.2 MESSAGE

The MESSAGE statement will write the specified field on the bottom line of the screen. Example:

```
<label>     MESSAGE field1
```

The bottom line of the form will be erased. However, the message is only temporary and the bottom line of the form will be restored when the operator writes the data record or erases the current record.

NOTE: The INPUT field is destroyed by the MESSAGE statement.

CODE: The MESSAGE generates 4 bytes of code: op, field1 length, field1 address.

3.4.3 SHOW

The SHOW statement will display a message in the current field area of the screen. If no message label is indicated, the SHOW statement defaults to the contents of the data record corresponding to the current field. Example:

```
           <label>     SHOW
or         <label1>    SHOW <label2>
```

The SHOW may be used if computations or table lookup conversions were made to change the value of the field. For example:

```
CRDRTAB     WORK "CREDIT","DEBIT "
LSTFLD      FIELD -1
CD          WORK "C","D"
MSG         WORK "  ";
S*          CONVERT    LSTFLD BY CD AND CRDRTAB
            GIVING MSG
            NEXT
```

Program 'S' is assigned to a keyin field (i.e. a field which reserves no data space) which is set to 'program reserve' (to automatically execute the program with no operator intervention). The program tests the preceeding field and displays a message corresponding to that value, for operator information.

NOTE: The INPUT field is destroyed by the SHOW statement.

CODE: The SHOW generates 3 bytes of code: op code, label2 address.

3.4.4 BEEP

When executed, the BEEP command causes the machine to issue a single beep sound. The format is simply:

```
           <label>     BEEP
```

CODE: The BEEP command generates only the 1 byte op code.

3.4.5 CHAIN

The CHAIN command enables the user to load a specific form when he exits his program. The format is:

```
        <label>    CHAIN n
```

where 'n' is the number of the form to be loaded (from 1 to 126 decimal, inclusive). The current data record is not written; however, the flag indicating data present is reset. The specified form is loaded and control is passed to the Interpreter at the first field of the new form.

CODE: The CHAIN generates 2 bytes of code: op, form number.

3.4.6 FORMSHOW

The FORMSHOW statement will cause the current form to be redisplayed. All data fields <u>on the screen</u> will be cleared. The output record is not affected and the current field index is not changed. Example:

```
        <label>    FORMSHOW
```

NOTE: INPUT is destroyed when FORMSHOW is executed.

CODE: The FORMSHOW command generates only the 1 byte op code.

3.5 Transfers of control

The three transfer of program control statements are the GOTO statement, the CALL statement, and the RETURN statement. The two field changing statements are the CHANGE and RESET statements.

3.5.1 GOTO

Control is immediately transferred to the label following the GOTO:

```
        <label>    GOTO label1
```

For the pre-defined labels, the word GOTO is optional. For programer defined labels, it is mandatory. Examples:

```
                   GOTO label1
                   GOTO NEXT
                   NEXT
```

CODE: The GOTO generates 3 bytes of code: op, label address.

3.5.2 CALL and RETURN

A single level of subroutine nesting is provided with the CALL and RETURN statements. A program may contain more than one set of CALL and RETURN statements -- but a CALLed subprogram may not CALL another subprogram. The statement formats are:

```
<label>     CALL <subprogramname>
<label>     RETURN
```

If a RETURN is executed with no preceeding CALL (in the current field program) a GOTO NEXT is executed.

CODE: The CALL generates 3 bytes of code: op, subprogram address. The RETURN generates only a 1 byte op code.

3.5.3 CHANGE and RESET

The transfer of data input control is the CHANGE statement. To change the input pointer from the current field (i.e., the sequence number of the field as it appears in the form) to another field, the new field number or displacement is specified immediately after the CHANGE verb:

```
<label>     CHANGE [sign]n
```

For example:

```
            CHANGE +1
```

After this instruction is executed, INPUT still contains the keyed-in data; however, the current field number has been incremented by one and OUTPUT now reflects the positions in the data record corresponding to the new field.

```
            CHANGE 1
```

After this instruction, however, the current field number has been changed to the first field in the form - field 1.

When the field program is entered the number of the current field is saved and may be restored at any time by executing a RESET command. This statement will reset the field pointers to the field current when the program was entered.

CODE: The CHANGE generates 3 bytes of code: op, flag, field number or displacement. The RESET generates only a 1 byte op code.

3.5.4 Pre-defined Labels

There are seven pre-defined labels which cause a transfer of control from the field program back to the DATAFORM Interpreter. These labels may be used as the destination address of comparison or GOTO instructions, as in the example:

```
          B*          IF NULL EQ INPUT THEN AGAIN
                      GOTO STORE
```

or may be referenced by name alone, as in:

```
          C*          ADD INPUT TO TOTAL
                      STORE
```

See the next section for detailed description of all pre-defined labels.

4.0 PRE-DEFINED LABELS

The seven pre-defined labels were first listed in section 1.1. They are discussed below.

4.1 AGAIN

This label passes control back to the DATAFORM Interpreter at a point which indicates an error to the operator and re-requests the current field.

4.2 CLOSE

This label passes control back to the DATAFORM Interpreter at the point corresponding to the operator typing the 'END' command. That is, an end of file marker is written on the data tape and the 'READY' message is displayed, leaving the operator in monitor mode.

4.3 END

This label passes control back to the DATAFORM Interpreter at the point corresponding to the operator depressing the write function key.

4.4 INPUT

This label designates the data keyed-in immediately prior to entering the field program. The data in INPUT has not yet been stored in the data record. It's length is defined at execution time by the length of the current field, and, it has been validated according to the edit criteria in the form itself prior to executing the field program.

4.5 NEXT

This label returns control to the Interpreter at the point at which the current field number is incremented. The cursor is moved to the next sequential field. (No data is stored.)

4.6 NULL

This label is a location which contains a binary zero. It may be used to determine, by testing the output record, if this is the first time data is entered. Example:

```
<label1>    IF NULL EQUAL OUTPUT THEN label2
```

Note that NULL should be referenced first since the length of the first operator is used for the comparison.

4.7 OUTPUT

This label designates the contents of the data record for the current field. If no data has been stored, OUTPUT has the value of binary zero (NULL). The length of OUTPUT is defined at execution time by the length of the current field. OUTPUT is undefined for ´keyin´ fields.

4.8 RETRY

This label is a location in the interpreter which contains a binary flag indicating whether the system is in modify or normal data entry mode. It can be checked by the user program by comparing it to NULL. If RETRY equals NULL the system is in normal data entry mode.

4.9 STORE

INPUT is <u>not</u> moved to the data record before control is transferred to a field program. The field program must do one of three things:

1) MOVE INPUT TO OUTPUT
2) MOVE somethingelse TO OUTPUT (where ´somethingelse´ may or may not be based upon INPUT)
3) exit the field program through the interpreter label STORE, which will automatically MOVE INPUT TO OUTPUT and position to the next field in the form.

5.0 PROGRAM GENERATION

Compilation of a program consists of two processes: using the editor to create a new source program, or edit an existing program; and compiling a new, newly edited, or old program.

5.1 Editing a source program

The Dataform 2 editor and compiler are resident on an LGO tape. The first file loaded is the editor, to permit editing or creating a tape to be input to the compiler. The Dataform editor is a special version of the General purpose editor; its command structure is that of the General purpose editor. The first display of the editor is:

```
CMP,OLD,NEW,DUP;PARAMETERS


?
```

A response of 'C' will begin the compilation of the cassette in the front deck with no editing of it. A response of 'O' will edit the information on the front cassette onto the rear cassette, and at the end of the editing process will copy the rear cassette onto the front. A response of 'N' will enter the new information onto the front cassette only. A response of 'D' will copy the front cassette onto the rear cassette and wait for a new cassette to be inserted in the front, and will then copy the rear cassette to the front. The 'PARAMETERS' field is discussed in the General Purpose Editor Manual. The 'Databus' option, which sets a tabstop at column 9, is initially set, and should be used for Dataform.

5.2 The compilation process

At the termination of the editing process, the compiler is loaded. The compiler displays:

```
DATAFORM 2 COMPILER 1.1
```

on the screen. If a printer is part of the compiling system, it asks if a listing is to be produced. It then reads through the front cassette preparing a symbol table, until it comes across a filemark. The front cassette is then rewound. The actual code generation and listing production takes place on the second pass over the input tape. The code produced is intermediately stored on the rear cassette, in the system scratch file (number 40). When the second pass over the front cassette is completed, the intermediately stored information on the rear cassette is transferred to the front cassette. At the completion of the transfer, some or all of of these messages will be displayed on the screen:

```
    STORAGE USED IN DECIMAL: 00000 RELOCATABLE, 00000 COMMON
    EXTENDED INTERPRETER REQUIRED
    FIELD PROGRAMS:
                A       00000
                Z       00000
    END OF COMPILATION:   NO ERRORS.
or  END OF COMPILATION:    n ERRORS.
```

These are descriptions of the program, telling the length of the entire program, whether or not the extended interpreter is required, and listing, in octal, the relocatable starting address of each of the programs defined. The End message lists the number of errors in decimal, if any occurred. After this message, the machine 'beeps', the keyboard and display lights are turned on, and the machine is set in an infinite loop. The compilation is then complete.

Any error messages are automatically displayed on the screen, with an asterisk indicating the part of the source line in error. The display may be stopped momentarily by depression of either the keyboard or display keys.

5.3 The compilation listing

The first act of the compiler is to test whether a servo or local printer is a part of the compiling system. If either of them are, the message:

```
            LIST ON SERVO PRINTER?
or          LIST ON LOCAL PRINTER?
```

is displayed. A response of 'Y' to this message will result in a printed listing of the program, as it is compiled. The listing consists of four parts:

1) the line number.
2) the initial address (either absolute or relocatable) associated with the statement line.
3) the code (in octal) generated for that line, eight bytes per printed line, using as many printed lines as necessary for the amount of code generated.
4) the line as it was input.

5.4 The Program Tape

When compilation is complete, the tape on the front deck contains two files: the source statements (file 0) and the compiled code (file 1). The compiled code file consists of a header record and both relocatable and absolute object code records.

The header record (which is in a fake object format) contains the number of the extended interpreter required (if any), the length of the relocatable object code, and the names and starting addresses of all field programs in the file.

5.5 Form Generation

The compiled program must be combined with a 'form' which contains the screen image, field definitions and program references (see Section II, 2). To combine the form and program files, load DF2FGS. If the form image has already been generated, place the form tape in the front deck and type 'OLD'; otherwise, generate a new form.

When the form is in memory, place a tape in the front deck and type 'OUT'. This will cause the program relocator overlay to be loaded. If programs have been specified for the form (during the PROG pass of the generator), the question:

```
DO YOU WANT TO USE A NEW PROGRAM TAPE
```

will appear. Type 'Y' and the message:

```
PLACE THE PROGRAM TAPE IN THE REAR DECK AND PRESS ENTER
```

will be displayed. Place a compiled program tape in the rear deck and press ENTER.

The header record is read to determine if this file contains any of the required programs. If it does, the entire file is copied to the front deck. The relocatable addresses at the beginning of each record are changed to absolute addresses by adding the address of the next free byte of user space (this will be different for each form).

If there are still unresolved program references, the new-program question will be repeated. If the user answers 'N' or all external references have been resolved, the form (pointers, image, edit table) is written on the front tape.

If an extended interpreter is required, the message:

```
     REPLACE DF2FGS AND PRESS ENTER
```

is displayed. Remove the program tape and put the system tape back in the rear deck and press ENTER. The required version of the extended interpreter will be copied to the end of the form.

When the form tape is completed, an end of file is written and the message:

```
     nnn BYTES LEFT
     DONE-LOAD NEXT SYSTEM
```

appears. Remove the form tape from the front deck.

All tapes should then be checked to see that they have appropriate labels on them, to avoid confusion later.

5.6 Testing

Place the DF2SYS tape in the rear deck and press RESTART. Place a blank tape (for data) in the front deck. When the Interpreter is running, type START to initialize the data tape. Then remove the DF2SYS tape and place the form tape in the rear deck. Type 'NEW' to cause the form to be loaded into memory. The form will be displayed with the cursor in the first data entry position. The user may then proceed to test the form. Undefined programs will not be executed, commands to automatically load other forms will be ignored in the test mode.

6.0 PROGRAM EXECUTION

6.1 Post process execution

Field programs are always executed as a ´post-process´ to data entry, that is, the program is not executed until the data has been keyed in and accepted by the basic interpreter. Thus, alpha/numeric checks, right justification, etc. will already have been performed on the input.

6.2 Operator tabbing

If the operator chooses to bypass a field which is not required, INPUT contains a null field (binary zeros).

If the cursor enters a field during backward or forward tabbing and no new data is entered, the data currently in the output record (which may or may not be a null field) is passed to the field program. If, however, new data is keyed in, the new data is presented to the field program in the INPUT area while previously entered data is still available in the OUTPUT area. If the previously entered data is cancelled by the operator, a null INPUT field is passed to the program.

6.3 Pre-process execution

To execute a ´pre-process´, the user must assign that pre-process program to the preceeding field (making certain to store or provide for storing the input data).

6.4 Program reserved fields

Data entry in a field may be prevented by designating that field as a ´program-reserved´ (P) field during the REQUIRED pass of form generation. In that case, the field program is executed immediately upon reaching the field and the area designated by INPUT is meaningless.

6.5 Form constants

Constants and semi-constants are set into the output area prior to data entry. However, fields containing constants will be passed through the basic interpreter as if the constant characters had been keyed in. They will be edited and passed to the field program in the INPUT area.

APPENDIX A: Sample programs

```
PARITIAL FORM

          NUMBER <^^^^^^  |^^^^^|
SIZE             7        6     1
TYPE             R
REQUIRED                  P     P
PROGRAM          S

MOVE SIGN TO RIGHT END OF FIELD

          INSIGN   REDEFINE INPUT,1,1
          INREST   REDEFINE INPUT,2,6
          NXTFLD   FIELD+1
          SIGN     FIELD+2
          SPACE    WORK " ";
          MINUS    WORK "-";
          .
          .        Input to 'keyin' field, move sign and store
          .        in next data field on form
          .
          S*       IF NULL NE INPUT THEN MOVE
                   IF NULL EQ NXTFLD THEN AGAIN
                   NEXT
          MOVE     MOVE INREST TO NXTFLD
                   MOVE INSIGN TO SIGN
                   IF MINUS EQ INSIGN THEN NEXT
                   MOVE SPACE TO SIGN
                   NEXT
          .
          .        Routine to do arithmetic with sign inverted field
          .
          TOTAL    WORK "00000.00"
          HOLDS    WORK "0";
          HOLDR    WORK "000.00"
          HOLD     REDEFINE HOLDS,1,7
          .
          .        Save keyed in total and zero computed total
          .
          A*       MOVE INPUT TO OUTPUT
                   SUB TOTAL FROM TOTAL
                   CHANGE 3
                   CALL ADD
                   CHANGE 6
                   CALL ADD
                   RESET
                   IF TOTAL NE OUTPUT THEN AGAIN
                   END
          .
          .        Routine to add sign inverted data
```

```
ADD         MOVE NXTFLD TO HOLDS
            MOVE OUTPUT TO HOLDR
            ADD HOLD TO TOTAL
            RETURN
```

MOD 10 CHECK DIGITS

```
         WEIGHT0  WORK "765432"
         CK0      IF INPUT CK10 WEIGHT0 THEN STORE
                  AGAIN
         WEIGHT1  WORK "21212"
         CK1      IF INPUT CK10 WEIGHT1 THEN STORE
                  AGAIN
         WEIGHT3  WORK "7137137"
         CK3      IF INPUT CK10 WEIGHT3 THEN STORE
                  AGAIN
```

MOD 11 CHECK DIGITS

```
         WEIGHT2  WORK "432765432"
         CK2      IF INPUT CK11 WEIGHT2 THEN STORE
                  AGAIN
         WEIGHT4  WORK "782345678"
         CK4      IF INPUT CK11 WEIGHT4 THEN STORE
                  AGAIN
```

COMPUTE REQUIRED CHECK DIGIT

```
         CKWORK   WORK "000000";
         CKDIG    WORK "0";
         CKIN     REDEFINE CKWORK,1,7
         1        WORK "1";
         WEIGHT   WORK "121212";
         C*       MOVE INPUT TO CKWORK
                  SUB CKDIG FROM CKDIG
         C1       IF CKIN CK10 WEIGHT THEN C2
                  ADD 1 TO CKDIG
                  GOTO C1
         C2       MOVE CKIN TO OUTPUT
                  NEXT
```

PARITAL FORM

```
          DATE |_|_|_   JULIAN |^^^^
SIZE           2 2 2           5
TYPE           D D D
REQUIRED       F F F           P
PROGRAM            J
```

JULIAN DATE CONVERSION

```
         JDAY     WORK "000031059090120151181212243273304334"
         DAYS     REDEFINE JDAY,1,3
         LEAPYR   WORK "7680848892"
         INMO     REDEFINE INPUT,1,2
         INDAY    REDEFINE INPUT,3,2
         INYR     REDEFINE INPUT,5,2
         JULIAN   FIELD +1
         JYR      FIELD +2
         K1       REDEFINE,JDAY,6,1
         K02      WORK "02";
         .
         .        Program starts here
         .
         .        use input  month as index to days/month table
         .
         J*       LOOKUP INMO IN DAYS GIVING JULIAN
         .
         . add input days
         .
                  ADD INDAY TO JULIAN
         .
         . check for leap year
         .
                  MOVE INYR TO JYR
                  IF INYR NIT LEAPYR THEN NEXT
                  IF INMO LE K02 THEN NEXT
                  ADD K1 TO JULIAN
         .
         . Julian date is in next field - it should be  'program reserve'
         .
                  NEXT
```

CHECK FOR DECIMAL FORMAT 000.00

```
         DEC32    REDEFINE INPUT,4,1
         DEC      WORK "."
         FLD32    WORK "000.00"
         .
         .        Use this if null field is permitted
         .
         OP32     IF DEC32 EQ NULL THEN FMATRET
         .
         .        Enter here if field is required
         .
         FMAT32   IF DEC32 NE DEC THEN AGAIN
         FMATRET  RETURN
         .
         .        If field in form is not initialized
         .        align input, store and redisplay
         .
         FMAT32   ALIGN INPUT TO FLD32
                  MOVE FLD32 TO OUTPUT
                  SHOW                    . Note that input must be saved
                  RETURN                  . before SHOW executed
         .
         .        If form contains semi-constant 000.00
         .
         FMAT32   ALIGN INPUT TO OUTPUT
                  RETURN
```

SHIFT LEFT and RIGHT

```
FLD      WORK 10
FLD1     REDEFINE FLD,1,1
LFLD     REDEFINE FLD,2,9
HOLD     WORK 9
FILL     WORK "0"
.
.        Shift left thru same field
.
SHFTLFT  MOVE LFLD TO FLD
         RETURN
.
.        Shift right thru hold field filling on left
.
SHFTRGT  MOVE FLD TO HOLD
         MOVE HOLD TO LFLD
         MOVE FILL TO FLD1
         RETURN
```

```
PARTIAL FORM "     "
        CREDIT/DEBIT <^^^^^^ |^^^^^
SIZE                 7       6
TYPE                 R
REQUIRED                     S
PROGRAM              M

MINUS OVERPUNCH

        FLD1     REDEFINE INPUT,2,6
        SIGN1    REDEFINE INPUT,1,1
        FLD2     REDEFINE INPUT,1,6
        SIGN2    REDEFINE INPUT,7,1
        .
        MOPCH1   WORK "0123456789"
        MOPCH2   WORK "{","J","K","L","M","N","O","P","Q","R"
        MINUS    WORK "-"
        KO       REDEFINE MOPCH1,1,1
        .
        .        Convert units position to minus overpunch using sign
        .        at left
        .
        .        input field should be 'keyin' (i.e. no output space reserved)
        .        store conversion in next field (should be program reserved)
        .
        NXTFLD   FIELD +1
        .
        . check for null field
        .
        M*       CALL NULLCK
        .
        . check for sign
        .
                 IF SIGN1 NE MINUS THEN PLUS
        .
        . convert units position to minus overpunch character
        .
                 CONVERT SIGN2 BY MOPCH1 AND MOPCH2 GIVING SIGN2
        PLUS     MOVE FLD1 TO NXTFLD
                 NEXT

CHECK FOR NULL FIELD ON INPUT

        .
        .        Check for null or repeated input on optional fields
        .
        NULLCK   IF NULL EQ INPUT THEN STORE
        SAMECK   IF INPUT EQ OUTPUT THEN NEXT
                 RETURN
```

CHARACTER CONVERSION FOR A FIELD

```
.
.        Character conversion for a field is accomplished
.        by shifting thru the field
.
IN1      REDEFINE INPUT,1,1
INMOV    REDEFINE INPUT,2,29
ASCII    WORK " 0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ!#"##$%&´()="
              "+*?/.,<>_-@;:"
EBCDIC   WORK 0100,360,361,362,363,364,365,366,367,370,371,301,302;
               303,304,305,306,307,310,311,321,322,323,324,325,326;
               327,330,331,342,343,344,345,346,347,350,351,132,177;
               173,133,154,120,175,115,135,176,116,134,157,000,113;
               153,114,156,155,140,174,136,172
CVTWK    DATA 1,30
WK1      REDEFINE CVTWK,2,29
WKSAV    REDEFINE CVTWK,30,1
K29      WORK "29"
COUNT    WORK "00"
K00      WORK "00"
.
C*       MOVE K29 TO COUNT                 . Initialize counter
C1       CALL CVTNAME                      . Convert 1 input char & store in
                                           .   units psn of output
         MOVE INMOV TO INPUT               . Shift input left
         MOVE W1 TO CVTWK                  . Shift output left
         SUB K1 FROM COUNT                 . Decrement counter
         IF K00 NE COUNT THEN C1           . at end convert last character
.
.        A RETURN will default to NEXT if the routine was not Called
.
CVTNAME  CONVERT IN1 BY ASCII AND EBCDIC GIVING WKSAV
         RETURN
```

BLANK FILL DISPLAY - ZERO FILL OUTPUT

```
.        Data field on form should have type ´B´ - right justified,
.        blank filled on the left
.
ZF       ADD NULL TO INPUT
         STORE
```

REFORMAT DATA RECORD

```
         5        FIELD 5
         EOL      WORK 015                  . END OF LINE ITEM
         EOR      WORK 03                   . END OF RECORD -  STORE THIS AFTER LAST 015
         DEC      WORK "."
         .
         .        After last field of line item, reserve a single position
         .        field and store an 015
         .
         B*       MOVE EOL TO OUTPUT
                  NEXT
         .
         .        At first field of line item, check for a special character
         .        e.g. a minus, a decimal point, a leading space, etc.
         .        If the special character is present, set an end of record
         .        into the field and force writing of the field
         .
         A*       IF DEC NE INPUT THEN STORE
                  MOVE EOR TO OUTPUT
                  END
         .
         .        OR write the record yourself and erase the line item data
         .        preserving the header information
         .
         HEADER   DATA 1,30
         REST     DATA 31,201
         .
         A*       IF DEC NE INPUT THEN STORE
                  MOVE EOR TO OUTPUT
                  WRITE
                  SET REST TO NULL
                  FORMSHOW
                  CHANGE 5
                  AGAIN
         .
         .        To elimanate unused positions in the data record
         .        shift the data left prior to writing
         .
         DATA1    DATA 29,199
         DATA2    DATA 30,200
         DATA3    DATA 57,198
         DATA4    DATA 58,199
         .
         .        NOTE that the data is no longer in the assigned positions
         .        and therefore may not be edited (MOD and FIND) by the original form
         .
         X*       MOVE EOL TO OUTPUT          . THE LAST FIELD IS A FAKE
                  MOVE DATA2 TO DATA1         . SHIFT END OF RECORD OVER 1ST UNUSED BYTE
                  MOVE DATA4 TO DATA3         . SHIFT AGAIN OVER 2ND UNUSED BYTE
                  END
         .
         .        Enter data out of order by using CHANGE instructions
```

```
PARTIAL FORM
         SHIPPED TO ¦^^^^^^^^^^^^^^          SHIPPED FROM ¦^^^^^^^^^^^^^^^
PROGRAM             A                                     A
         ADDRESS    ¦^^^^^^^^^^^^^^          ADDRESS      ¦^^^^^^^^^^^^^^^
PROGRAM             A                                     B
         DATE       ¦^¦^¦^                   DATE         ¦^¦^¦^
PROGRAM                 C

ENTERING FIELDS OUT OF ORDER
         NXTFLD   FIELD+1
         FLD2     FIELD 1
         FLD8     FIELD 7
         A*       CALL SAVE
                  CHANGE +1
                  NEXT
         B*       CALL SAVE
                  CHANGE 8
                  NEXT
         C*       CALL SAVE
                  CHANGE 2
                  NEXT
         SAVE     MOVE INPUT TO OUTPUT
                  RETURN
```

TOTAL ACCUMULATION

```
         TOTAL    COMMON "00000.00"
         A*       SUB OUTPUT FROM TOTAL
                  ADD INPUT TO TOTAL
                  STORE
         .
         .        or
         .
         B*       SUB OUTPUT FROM TOTAL
                  ADD INPUT TO TOTAL
                  MOVE INPUT TO OUTPUT
                  MESSAGE TOTAL                   . Display current value of
                  NEXT                            .  the accumulator
         .
         .        For Intra-form totals
         .
         1        FIELD 1
         C*       ADD FLD1 TO FLD2 GIVING TOTAL
                  ADD FLD3 TO TOTAL
                  ADD FLD4 TO TOTAL
                  IF TOTAL EQ FLD5 THEN END
                  CHANGE 1
                  AGAIN
         .
         .        For line-item totals
         .        use definitions like these
         .
         FLD1     FIELD -5
         FLD2     FIELD -4
         FLD3     FIELD -3
         FLD4     FIELD -2
         FLD5     FIELD -1
```

```
PARTIAL FORM
         NAME |^^^^^^^^^^^^^ DATE |^|^|^  ACCOUNT |^^^^^^ <
REQUIRED                          P P P                  P
PROGRAM        V                                 V       L

MODIFY MODE VERIFY PROGRAM

         0        WORK "0";
         1        WORK "1";
         RETRY    WORK "3";
         HOLD     WORK "              "
         N        WORK "0";
         .
         .  VERIFY PROGRAM
         .
         V*       IF INPUT EQ OUTPUT THEN OK
                  ADD 1 TO N
                  IF N EQ RETRY THEN CKHOLD
                  MOVE INPUT TO HOLD
                  AGAIN
         CKHOLD   IF INPUT NE HOLD THEN VAGAIN
                  MOVE INPUT TO OUTPUT
         OK       SUB N FROM N
                  NEXT
         .
         VAGAIN   SUB N FROM N
                  AGAIN
         .
         .  LAST FIELD PROGRAM REWRITES RECORD
         .  (IN MODIFY MODE NEXT RECORD AUTOMATICALLY READ)
         .
         L*       END
```

APPENDIX B: COMPILER ERROR MESSAGES

PARITY ERROR: A/C?

A parity error persisted on a cassette read operation after five retries. A response of 'A' will abort the compilation; a response of 'C' will use the bad block as if nothing were wrong with it.

BAD LABEL INITIATIOR

A character that was neither a decimal point nor a space nor alphanumeric appeared in column 1 of the input line.

INVALID OCTAL

The character string pointed to by the asterisk contains a character which is not in the set 0-7.

ILLEGAL OPERATOR

Something other than those operators appearing in APPENDIX G was the first nonblank symbol after column 1 (or after the label, if one exists).

NUMBER FROM 1-245 EXPECTED

The indicated symbol is nun-numeric, or if numeric, not in the specified range.

COMMA EXPECTED

The symbol after the first number in a DATA statement was not a comma.

FIELD2 IS LESS THAN FIELD1

In a DATA statement, the second field is less than the first.

LABEL REQUIRED

The DATA, REDEFINE and WORK statements all require a label.

DOUBLE QUOTE ASSUMED

A pre-defined constant (either in WORK or COMMON statements) should be terminated by a double quotation mark. If it is not there, it is assumed.

ILLEGAL LITERAL

In a table, every item enclosed in double quotation marks must be of equal length. Those that are of different length than the first item are flagged in error.

IMPROPER CONTINUATION

If a COMMON or WORK table is continued from a line, the following line must have a blank in column one, and the first symbol on the line must be a double quotation mark. If either of these is not the case, the continuation is an improper one.

UNDEFINED LABEL

A label is referenced which is neither one of the eight pre-defined labels, nor defined elsewhere in the program.

MISSPELLED WORD

A specific reserved word -- for example, the TO in an ADD statement -- has been misspelled. The misspelled word is assumed to be the one expected, and the next symbol is expected to be a legal label.

ILLEGAL CONDITION

The connective in an IF statement is not one of those listed in section 3.3. Nothing about the connective is assumed.

DUPLICATE LABEL

The label beginning the line listed is duplicated previously in the program (or it is one of the eight pre-defined labels). The second (and any subsequent) definitions of the label are ignored.

MAXIMUM LABELS REACHED

The maximum number of labels allowed by the compiler is fixed at 125, including the pre-defined labels. All labels after this maximum is reached are ignored.

COMMON LIMIT EXCEEDED

The COMMON block may not exceed 100 bytes. Anything defined as COMMON after this length will not be accepted.

PROGRAM COUNTER ERROR

The program counter, at the end of pass two does not equal the program counter at the end of pass one. This is an internal compiler error message.

APPENDIX C: COMMANDS REQUIRING THE EXTENDED INTERPRETER

ROUTINE	DECIMAL SIZE
MULTIPLY	83*
DIVIDE	183*
CONVERT	83
CK10 & CK11	161
*MUL/DIV OVERHEAD	56
EXTENDED INTERPRETER OVERHEAD	18
COMMON (REQUIRED WITH EXTENDED INTERPRETER)	100

APPENDIX D: Code Reduction Techniques

1. Use carets (^) in field definitions (remember they are compressed while dashes are not).

2. Use 'common' instead of 'work' if any extended interpreter is used (100 bytes of common is reserved whether you use it or not).

3. Place semi-colons at the end of all non-table, non-range variables to suppress the end-of-table character.

4. Use 'redefine' to create constants or tables which are subsets of other constants or tables. This technique may also be used for computation or hold areas if the redefined variables are not needed at the same time.

5. Use subroutines to perform repeated operations.

6. Use field displacement referencing to generalize programs used with line-items (i.e., where the same set of fields is entered several times within one form).

7. Use 'input', 'output' and 'reset' to generalize programs and thus avoid duplication of code.

8. Keep constants in the form itself (by defining them at form generation time) instead of using a field program to set them.

9. Combine several fields into one wherever possible (each field requires 6 additional bytes of edit table).

10. Avoid extended interpreter functions when possible (by coding multiplies using add's, etc.).

11. Use 'Lookup' instead of 'Convert' to save one of the tables.

12. Use data areas as work areas whenever possible, thus saving intermediate hold areas.

13. Execute all programs on last field if possible, on the assumption that the operator is usually right, to save 'NEXT' and 'STORE' instructions.

14. Don't use CHANGE/SHOW/CHANGE instructions if NEXT will automatically show and bypass if field is defined as 'program reserved'.

APPENDIX E: ALPHABETICAL LISTING OF STATEMENT TYPES

NAME	OPCODE	SECTION
ADD	0117	12
AGAIN		19
ALIGN	0113	9
BEEP	0137	15
CALL	0124	17
CHAIN	0130	16
CHANGE	0126	17
CLOSE		19
COMMON		6
CONVERT	0116	10
DATA		3
DIVIDE	0122	12
END		19
EQU		6
FORMSHOW	0134	16
FIELD		8
GOTO	0123	16
IF CK10	0135	13
IF CK11	0136	13
IF INT	0100	13
IF NIT	0101	13
IF INR	0102	13
IF NIR	0103	13
IF EQ	0104	13
IF NE	0105	13
IF GE	0106	13
IF LE	0107	13
IF GREATER	0110	13
IF LESS	0111	13
INPUT		19
LOOKUP	0115	10
MESSAGE	0131	14
MOVE	0112	11
MULTIPLY	0121	12
NEXT		19
NULL		19
OUTPUT		20
REDEFINE		7
RESET	0127	17
RETRY		20
RETURN	0125	17
SET	0114	11
SHOW	0133	15
STORE		20
SUBTRACT	0120	12
WORK		3
WRITE	0132	14

SECTION III
Programmers Manual

Table of Contents
Section III

1.0 System Structure of the Interpreter 1

2.0 Edit Table Format 2
- 2.1 Format 2
- 2.2 Work Area 3
- 2.3 Routines to Access Edit Table 3

3.0 Structure of a Form in Memory 4
- 3.1 Pointers 4
- 3.2 Data Buffers 4
- 3.3 Form Image 4
- 3.4 Edit Table 5
- 3.5 Field Programs 5
- 3.6 Extended Interpreter 5

4.0 Subroutines Available in the Interpreter 6
- 4.1 Interrupt Handler 6
- 4.2 Cassette Drivers 6
- 4.3 Keyboard Input 6
- 4.4 CRT Display 6
- 4.5 Form and Data Access 7
- 4.6 String Arithmetic 7

5.0 T8
- 5.1 Program 8
- 5.2 Form 8
- 5.3 Data 9

6.0 10s
- 6.1 External References 10
- 6.2 Interpreter Processing Addresses 10
- 6.3 Interpreter Data Areas 10
- 6.4 Loading Assembly Language Programs 11

1.0 System Structure of the Interpreter

The Interpreter resides within an 8K Datapoint 2200. Of the 8K the first 6.5K is reserved for the Interpreter. The remainder is shared by the user's data area, edit tables, form image, and, if necessary, field programs (which, in turn, may required the extended interpreter). In the Interpreter, memory is allocated as follows:

subroutines*	00000-00777
variable data	01000-01643
interrupt handler	01644-01751
cassette drivers	01752-04717
keyboard I/O	04720-06272
command handler	06373-12142
field program interpreter	12144-13421
string arithmetic	13422-14523
form pointers	14524-14740
user space*	14741-17777

* Some Interpreter subroutines are initially set in the user space area and are moved into low memory when the Interpreter is first executed.

2.0 Edit Table Format

2.1 Format

For each field defined by a form, a six byte edit table entry is generated. The entry contains:

- horizontal position
- vertical positon
- length of field
- position in output record
- edit key
- field program letter

The horizontal position (0-79) indicates the starting column of the field in the screen image. The vertical position (0-11) indicates the line of the screen image containing the field. The information is used to display the field as well as to access data stored in the form image for the field (i.e., constants).

The length of the field is the number of characters the operator may key in (1-80). This number is associated at execution time with the labels INPUT, OUTPUT and with field references in user field programs.

The position of the field in the data record is actually an index (0-244) into the output buffer. If the field is a 'keyin' field, i.e., no data space is reserved, the position value is 0377.

The edit key is a combination of bits indicating the edits set in the Generator passes TYPE and REQUIRED. The bits in the edit key have the following meanings:

```
 _______________________________
|7 | 6 | 5 | 4 | 3 | 2 | 1 | 0|
 _______________________________
  \   \   \   \   \   \   \   \___Alpha
   \   \   \   \   \   \   \_____Numeric Field
    \   \   \   \   \   \_______No Keyin
     \   \   \   \   \_________Right Justified
      \   \   \   \___________Zero Fill
       \   \   \_____________Numeric Digits
        \   \_______________Fill Controlled
         \_________________Required
```

The alpha and numeric digit bits are both set for the 'mixed' field type.

The field program letter is set to binary zero if no field program is assigned; otherwise, the actual ASCII letter is stored in this byte.

The number of the last field in the edit table (the first field is zero) is used to determine the length of the table. In addition, there is an 0377 stored after the last entry.

2.2 Work Area

During data entry, the edit table entry for the current field is moved to a work area in the data page for ease of referencing. The variables:

```
COLUMN
LINE
LENGTH
PSN
EDTKEY
USER
```

contain the six byte entry. 'SAVFLD' contains the current field number.

2.3 Routines to Access Edit Table

There are several routines available to access the edit table entries. EDTPNT is the most basic routine. This routine uses the value in the C-register to set the HL registers to the address of the corresponding field.

MOVEDT saves the field number in the C-register in SAVFLD, and moves the corresponding edit table entry to the work area and into the registers. It also positions the cursor to the field.

NEXT and LAST use the saved field number to access the next or the preceeding field. Both routines call MOVEDT.

3.0 Structure of Form in Memory

3.1 Pointers

The form is defined by a fixed set of pointers:

linked form number
field program pointers
maximum field number
edit table pointer
data-write buffer pointer
length of data record
form line pointers

The variable NEXTF contains the number of the linked form (000 if no link, otherwise it is the physical file number of the form), and the variable PAGE3 is the auto-link flag (0 or 0377).

For each possible field program four bytes are reserved starting at the label USERA. The four bytes are zero if the corresponding program letter is not present. If a program is present, whether referenced or not, the first pair of bytes contains the 'base address' to be used for all relative addresses within the field program. The second pair of bytes contains the starting address of the program. (Note: All addresses are stored MSB,LSB.) Unresolved program references contain an octal 377 in the first byte.

The edit table is always referenced via the address pointer, SEDIT, and the maximum field number, EEDIT, which is checked against the requested field.

3.2 Data Buffers

The input data buffer is always in a fixed position, DATA, at the end of all form pointers. It's length is define by the variable LDATA.

The output buffer, to which the data is moved prior to writing, is in a variable position. It is set at the end of the data buffer, at a point defined by the length of the data record + 8. The address of the output buffer is in SMATCH. The output buffer is also used when performing FIND operations. The data contained in the output buffer is available to the operator by means of the field dup function key (DISPLAY/D).

3.3 Form Image

The compressed form is stored beyond the two data buffers and it is referenced indirectly through the pointers starting at LINES. If the address in the table of pointers starting at LINES, corresponding to one of the twelve screen lines, is zero, the corresponding line is to be blank on the screen.

3.4 Edit Table

The edit table is generated beyond the compressed form. The byte immediately after the edit table terminator (0377) is available for user programs.

3.5 Field Programs

When programs are attached to the form, using the Field Program Relocator, blocks starting at relocatable addresses are giving absolute addresses based on at the first available space after the form edit table. Non-relocatable records from the field program (e.g. COMMON), are simply passed through to the form tape.

3.6 Extended Interpreter

There are fifteen extended interpreters which contain all possible combinations of four extended commands (convert and lookup, and CK10 and CK11 are combined for purposes of the extended interpreter). Thus, extended interpreter 1 contains only check digits, 2 contains the multiply subroutine, 3 both check digits and multiply, 4 divide, 5 divide and check digit, 6 multiply and divide, 7 multiply, divide and check digits, 8 conversions, 9 conversions and check digits, 10 conversions and multiply, 11 conversions, multiply and check digits, 12 conversions and divide, 13 conversions, divide and check digits, 14 conversions, multiply and divide, and 15 contains all extended functions.

The extended interpreters are all assembled so that they end 100 bytes plus 3 to 18 bytes (for jump instructions) from the end of memory (in an 8K machine); thus leaving a maximum amount of user space. Three to eighteen of the bytes are reserved for a jump table into the extended interpreter itself, since the starting addresses of the subroutines change for each of the fifteen levels of interpreter.

4.0 Subroutines Available in the Interpreter

4.1 Interrupt Handler

The Dataform interrupt handler, INTRPT, is labelled and works like the D.O.S. interrupt handler. There is, however, room for only two interrupt processes. Process 1 is reserved for the cassette driver.

4.2 Cassette Drivers

The names and parameterization of the cassette routines in Dataform are the same as for the D.O.S. cassette routines. There are two additional routines available: TRAW$ and LOAD$.

TRAW$ is a tape write routine which automatically performs a read-after-write function. It is used to write the data records and is parameterized as is TWRIT$.

LOAD$ will load object code (forms are in object code format) at the addresses specified by each record. The tape must be positioned past the file marker before calling LOAD$. The only parameter is the deck in the B register. The routine loads continuously until a file marker is encountered. The tape is left positioned beyond the file marker record and the variable NXTFIL contains the number in that file marker record. LOAD$ returns as soon as the load operation has been initiated. Thus, the user must call TWAIT$ to determine when the load has been completed.

4.3 Keyboard Input Routine

There are two entry points to the keyboard input routine, KEYIN and KEYIN$. When the routine is entered at KEYIN, the edit type and length for the current field are applied to the input. In addition, it is assumed that the corresponding area of the form image is in the HL registers. This area is checked for constants. If entered at KEYIN$, fake parameters are provided to permit twenty characters with no edit restrictions. <u>The input is always stored in TEMP</u>.

4.4 CRT Display Routine

The display routine also has two entry points, DSPLY$ and DSPLY. If entered at DSPLY, the cursor position will be set to the bottom line and the screen will be rolled up after the message is displayed (all messages must be terminated by an 015). If entered at DSPLY$, the contents of DE will be used to position the cursor and no rollup will take place at the end of the display.

There are two special characters permitted in the display input message: 023, which may only appear at the beginning of the message, cause the screen to rollup one line, and 011, which may appear anywhere in the message, followed by a count indicates space compression. In addition, binary zeros are converted to underscores and spaces are <u>not</u> displayed at all (i.e., the cursor is simply positioned to the right).

The routine called REWRT redisplays the form (with no data).

4.5 Form and Data Access Routines

The routine GETADR uses the contents of the variables HP and VP to locate to positions in the form image corresponding to the current field (this is where constants and semi-constants are stored).

GETDAT sets HL to the address in the data buffer corresponding to the current field. The B-register contains the length of the field.

MOVEDT uses the value in the C-register to access the edit table entry corresponding to that field and moves the six byte entry to a work area for easy referencing. It also saves the field number in the variable SAVFLD.

4.6 String Arithmetic Package

The string arithmetic package used in Dataform requires the following parameters:

HL = destination and field operated on
DE = operator (i.e., divisor)
the length of HL is in BLEN
the length of DE is in ALEN

Only the add and subtract functions are available in the basic interpreter. The addresses of multiply and divide change depending on the particular level of extended interpreter being used.

The entry point for add is ADD$ and for subtract is SUB$.

5.0 TAPE FORMATS

All Dataform tapes are written according to Datapoint standard formats. File marker records, record types and lengths conform to the Serial Numeric tape format and internal structure of the records is based on GEDIT (for text) and ASM (for program and form) formats.

Tapes may be IN'd to disk and processed; however, data records OUT'd from disk will no longer have the necessary form number at the end.

5.1 PROGRAM

```
______________________________________________________________________
|FM|Source Statements in |FM| Header | Relocatable and Absolute |FM |
|0 |GEDIT compressed fmt |1 |  (obj) |  object code format      |127|
----------------------------------------------------------------------
                                 |             |
                                 |             |
                                 |             |____Standard object
                                 |             format:     H/L/-H/-L/
                                 |             Relocatable addresses
                                 |             in MSB.  Absolute and
                                 |             relocatable not mixed
                                 |             withing blocks.
                                 |
 ____________________________________________________________________
 |000|000|377|377|C-MSB|C-LSB|flag|Prog|MSB|LSB|. . .|Prog|MSB|LSB|0|
 --------------------------------------------------------------------
          |           |       |          |                         |
 Fake starting    Length of   |    Program names and      Terminator
   address of    relocatable  |       addresses
     block           code     |
                              |
                   Extended interpreter
                         number
```

5.2 FORM

```
_______________________________________________________________________
|FM|FM| O P T I O N A L |                       | O P T I O N A L |FM |
|0 |1 | User programs   | Form pointers, image  |  Extended       |127|
|  |  | and common      | and edit tables       |  Interpreter    |   |
-----------------------------------------------------------------------
```

5.3 DATA

```
 _____________________________________________________________
|FM | Data records - to length defined by form | FM  | FM  |
|0  |                                          |127  |127  |
--------------------------------------------------------------
                         |
   _________________________________________________________________
  |Record type |Data |Data . . . |Data |015 |003 |Form    |Rewrite|
  |and Parity  |     |           |     |    |    |Number  |Counter|
  -------------------------------------------------------------------
       |                                   |    |      |        |
   Serial Numeric                          |  Physical |    Number of
    format                                 |   end     |      Rewrite
                                           |           |      changes
                                           |           |    (1-ASCII)
                                      Logical      Number of
                                        end       controlling
                                                 form (1-binary)
```

6.0 Assembly language interfacing and overlays

6.1 External References

Facilities are provided in the Dataform language to reference points outside the program, locations which may be either in the Interpreter itself or in an assembly language program separately assembled by the user.

The EQU instruction assigns an address to a label which may then be referenced by any of the branching statements in Dataform (GOTO, CALL, etc.). If this facility is used, it is up to the user to return control to the proper point in the Intepreter or the field program.

6.2 Interpreter Processing Addresses

A jump table of Interpreter entry points is provided so that these address will not change in future versions:

```
        NEXT$           EQU   01147
        AGAIN$          EQU   01152
        STORE$          EQU   01155
        END$            EQU   01160
        ERASE           EQU   01166
        WEOF$           EQU   01163
        RETURN$         EQU   013014
```

To return to a field program after being called, the assembly language program should jump to RETURN$. Otherwise, a jump to the appropriate exit routine will return control to the Interpreter.

6.3 Interpreter Data Areas

Various Interpreter data areas may be needed by the user programs. The variable TEMP is the keyin buffer and it is this area which is accessed when 'INPUT' is referenced in a field program. INPUT is compiled as an address of 01000 and a length of zero. At execution time, the length of the current field is substituted. OUTPUT, compiled as address zero and length zero, is resolved at execution time. It is converted to the length and address in the data buffer of the current field.

Labels defined in FIELD statements are compiled with lengths of one and a special code in the MSB portion of the address. If the MSB is 0370, the LSB represents an index to the field table (i.e. the field number supplied by the programmer, minus one). If the MSB is 0375, the LSB represents a displacement which, at execution time, is added to the current field number in order to resolve the length and address information.

Note that referencing a field other than the current field does not change the number of the current field.

6.4 Loading the Assembly Language Program

Since the format of a form tape and that of assembly code is the same, an assembly language overlay may be loaded by using the 'NEW' command of the Interpreter. The user should first load the form using 'NEW' and then the overlay.

Once the form and program have been tested, there are several ways to put the system together:

1) The assembly program may be cataloged as a separate form and be loaded by either the operator or by a field program.

2) The form and the assembly language program may be appended together using the facilities of either the Cassette Tape or Disk Operating System.

3) The assembly language program may be appended to the Intepreter (again using the tape or disk operating system) so that it is always available. This should be done only if the program is never overlayed by anything else. The user must be careful to insure that the appended program contains the proper transfer address for the Interpreter.