DISKETTE DATA ENTRY SYSTEM DF11SYS User's Guide

Version 2

March, 1976

Model Code No. 50135

DATAPOINT CORPORATION

The Leader in Dispersed Data Processing

PREFACE

DF11 provides a personalized data entry system for use on any DATAPOINT processor and diskette. DF11 is intended primarily to support generation of data entry systems on diskette. DF11 contains more features and greater capabilities than DATAFORM 2 contained.

Forms" are created for display on the processor's screen, and the data entry operator then simply fills in the form. The data is then recorded, and at any time may be retrieved and revised using the same form to view and edit the recorded data.

Each "form" is custom designed, and editing criteria are assigned to the data fields on the form at the time the form is generated. Programs written in the high level DF11 language may also be assigned at this time. Forms and programs are then combined and become a unique DF11 system.

Three stages of development are involved in generating a system: the editor and compiler are used to create field programs; the form generator is used to create forms; and the data entry interpreter is used to control data entry. Additionally, the print utility provides a hard copy listing of the data file, form images, or both

Since DF11 uses standardized data record formats, further processing of the data can proceed under any DATABUS, BASIC, or RPG program. Additionally, any one of a number of available communications programs or terminal emulators (including DATAPOLL and EM2780) may be used to transmit DF11 data files for further processing at remote sites.

Chapter 1 should provide as much information about DF11 as is necessary for those familiar with both the cassette version of DATAFORM and DOS. Chapter 2 provides a more general description of DF11, and continuing chapters describe forms generation and data entry using the forms. The DOS User's Guide, model number 50127, provides more information about the DOS.

Throughout this manual, a field appearing between pointed brackets, as:

    <filename>

denotes a required field; whereas, a field appearing between square brackets, as:

    [,filename]

denotes an optional field, whose use is explained in subsequent discussion.

Version 2 of DF11 contains the following changes from version 1 of DF11:

1) The execution of the CLOSE statement has changed.
2) The S type edit criteria has been added to suppress shift key inversion.
3) The interpreter's START command displays the name of the data file.
4) The default data file extension is TXT.
5) Keyin over-run is prevented.
6) Data file OVERFLOW action has changed.
7) DF11 interpreter now uses only 251 characters in a

disk sector, rather than 253. This means that the version 1 interpreter must be used with version 1 data files; and the version 2 interpreter must be used with version 2 data files.

8) The HELP command is no longer in the form generator.

To convert DF11 version 1 systems to DF11 version 2 systems, each field program should be re-compiled, and each form should be re-generated.

TABLE OF CONTENTS

CHAPTER 1. A QUICK GUIDE TO THE DISKETTE DATAFORM SYSTEM

1.1 Installing the Diskette DATAFORM System

DF11 is released on a flexible diskette, listed in the software catalog as DF11SYS.

The following files are included on the DF11SYS release diskette:

DF11CMP/CMD	DF11 program compiler
DF11GEN/CMD	DF11 form generator
DF11PRT/CMD	DF11 print utility
DF11/CMD	DF11 interpreter
DF11/OV1	"HELP" overlay
COPYDF/TXT	chain file to copy the DF11 system

Additionally, the following DOS.C commands are necessary. They should be obtained from the latest release of DOS.C.

```
CHAIN/CMD
CHAIN/OV1
COPY/CMD
EDIT/CMD
```

Immediately upon receipt of the DF11SYS release diskette, several copies should be made for backup purposes. This is accomplished by placing the release diskette in drive zero, placing a DOS.C system diskette in drive one, placing a DOSGEN'ed diskette in drive two, and entering:

```
CHAIN COPYDF;TO#2#
```

1.2 System Names

DF11 utilizes a concept called a "System Name" (which is abbreviated "SYSNAM"). SYSNAM is a one to six character alphabetic name. All forms in a system, i.e. forms that are to be used together, should be assigned the same system name followed by a 2 digit number. Programs to be used with a particular form should be assigned the same system name and number as the form. The program source file (as created by EDIT) will have an extension of "TXT", the compiled program object file (as created by DF11CMP) will have an extension of "DFP", and the form (as created by DF11GEN) will have an extension of "DFF" (SYSNAMnn/DFF). The initial data file (as created by DF11) will have the name and extension "SYSNAM/TXT".

The file extensions mentioned above are created and maintained by the particular program being run (EDIT, DF11CMP, DF11GEN, and DF11). They should not be changed.

1.3 Program Generation

1.3.1 Program Source File Creation

To generate a program enter:

```
EDIT <SYSNAMnn>;D
```

where "SYSNAM" is the name of the system of forms and "nn" is the 2 digit number of the form with which the program(s) will be used. A file named SYSNAMnn/TXT will be created. The ";D" on the command line causes DF11 tab stops to be used.

When all program statements have been entered, and the EDIT has been terminated by use of the ":E" command, the program statements are recorded, and the DOS is reloaded. See the chapter on EDIT in the DOS User's Guide for EDIT commands and further EDIT parameterization.

1.3.2 Program Compilation

To compile a program, enter:

```
DF11CMP <SYSNAMnn>[,objectfile][;parameters]
```

The compiler identifies itself with the sign on message:

```
DF11 COMPILER 2.n - ddmmmyy
```

The compiled object code is placed in the [objectfile]. The default [objectfile] name is the same as the name of the source file. The default [objectfile] extension is "DFP". Parameters may be entered at the time the compiler is executed. The parameters are separated from the [objectfile] name (if a name is present) by a semi-colon. If only a semi-colon is entered, the compiler assumes that no listing is to be printed. If the letter "L" appears after the semicolon, a listing without code will be generated. If the letters "L" and "C" both appear after the semicolon, a listing with code will be generated. If the letter "P" appears after the semicolon, a printer image file will be generated on the disk. If "P" and "C" appear, generated code will be included in the printer image file. The printer image file will be given the name "SYSNAMnn/PRT". This file may be printed or viewed on the screen with the DOS LIST utility. See the chapter on LIST in the DOS User's Guide for LIST parameterization.

If no parameters are entered, and a printer is on line, the messages:

```
LIST ON LOCAL/SERVO PRINTER?
```

and

```
LIST CODE TOO?
```

must be answered.

1.4 Form Generation

To generate a form enter:

```
DF11GEN <inputform>[,outputform][,objectprogram]
```

The generator identifies itself with the sign-on message:

```
DF11 GENERATOR 2.n - ddmmmyy
```

The <inputform> name must be in "SYSNAMnn" format. The default [,outputform] name is the same as the <inputform> name. The default [,objectprogram] name is the same as the [outputform] name, but with a default extension of "DFP".

DF11GEN responds to the following commands. Most are the same as the cassette form generator's.

```
CONSTANT
JUSTIFY
LINK
NEW
OLD
OS
OUT
PROGRAM
REQUIRE
REVISE
SEMI-CONSTANT
TYPE
```

When "OUT" is entered, if field programs are assigned, the entire [objectprogram] file will be attached to the [outputform] file. If the [outputform] name is different from the <inputform> name, the [outputform] name should appear on the command line. Thus, when "OLD" is entered, <inputform> will be read; and when "OUT" is entered, the form in memory will be written to [outputform].

Entering "OUT" automatically reloads the DOS if the form generation is successful. The DOS may be reloaded at any time without writing the form by entering "OS".

1.5 Form Testing and Data Entry

To test the completed form, or to perform data entry, enter:

```
DF11 <SYSNAMnn>[,datafile][;mode]
```

The interpreter identifies itself with the sign-on message:

```
DF11 INTERPRETER 2.n - ddmmmyy
```

The default extension of the data file is "TXT". If no [datafile] name is entered, the default data file name is "SYSNAM/TXT". Form "SYSNAMnn" is loaded, the [datafile] is opened.

The initial data entry mode may be specified on the command line, by placing the first character of the mode ("S" for START; "A" for ADD; "F" for FIND; and "M" for MODIFY) after a semicolon. The commands available in the DF11 interpreter are:

```
ADD
BACKSPACE
CONTINUE
DATA
END
FIND
HELP
LOAD
MODIFY
OS
REWIND
START
```

The "START", "ADD", "MODIFY", or "FIND" commands initiate interpreter action. Execution is the same as for the cassette interpreter with the exception that entering "END" both terminates the data file and reloads the DOS.

The "OS" command reloads the DOS without terminating the data file.

To maximize the space available for data entry, the data file should be created on a drive other than the drive containing the DF11 system's diskette.

1.6 Using DF11 with CHAIN

The DOS CHAIN command is a versatile system for providing a control sequence which is virtually operator proof. It can also be useful during development for repeated re-compilation, form generation and test sequences. See the chapter on CHAIN in the DOS User's Guide for CHAIN commands and parameters.

All programs in the DF11 system may be accessed via a CHAIN file. However, not all types of operator input may be provided by that file. For the form generator, all commands may be entered but none of the characters required by the various commands may be entered. For the compiler and print utility, all keyin requests, answers to questions, headings, etc., may be supplied by the chain file. No keyins may be provided for the interpreter (other than the initial command line); however, when data entry is complete, the "OS" command causes the CHAIN to resume.

The following is and example of a developmental CHAIN file for a system called BNK:

```
DF11CMP BNK01
DF11CMP BNK02;LC
TELLER TERMINAL REPORT - #DATE#
DF11GEN BNK01
OLD
OUT
DF11GEN BNK02
OLD
OUT
DF11 BNK;S
```

The following is an example of a CHAIN file for data entry:

```
//* PLACE DATA DISKETTE IN DRIVE 1 - PRESS DISPLAY KEY
DF11 BNK,:DR1;S
LIST BNK;LX
TRANSACTION REGISTER #DATE#
SORT BNK,BNK/SRT;1-10
LIST BNK/SRT;LX
SORTED TRANSACTION REGISTER #DATE#
//
//TRANSMISSION PHASE
//
DPDMP
690-7543
1
BNK/TXT
**
**
//TRANSMISSION COMPLETE
```

1.7 DF11 Compatibility

DF11 runs in a 16k cassetteless 1100 with a minimum of 1 diskette drive. It has all features of the DATAFORM 2 interpreter and form generator, and selected features of the configurator. DATAFORM 2 form images may be converted to DF11 form images, and DF11 form images may be converted to DATAFORM 2 form images by using the separately released program DFCONV. DATAFORM 2 source code programs (with no assembler routines or EQU's to interpreter data or routines) are compatible with DF11 source code programs with the sole exception of the CHAIN statements.

The following changes have been made to DATAFORM 2 to produce DF11, and are due mainly to the use of a new media and a larger machine:

1. User space has been increased to 5k.
2. Forms load from diskette in less than a second.
3. The extended interpreter overlays of DATAFORM 2 are resident in memory.
4. Data records are packed into and span disk sectors.
5. DF11 is CHAIN compatible.
6. Logical record size has been increased to 249 characters.

The following expansions have been made to the DATAFORM 2 language to produce the DF11 language:

1. Variable amount of common.
   (RESERVE nn)
2. Program control over data entry mode.
   (ENTRYMODE, MODIFYMODE)
3. Program controlled end of file on data file.
   (WEOF)
4. Program controlled backspace on data file.
   (BACKSPACE <LABEL>)
5. Program controlled position to end of data file.
   (PEOF)
6. Program controlled record deletion.
   (DELETE)
7. Program controlled record read.
   (READ)
8. Interrogation of current field number.
   (FIELDNO <label>)
9. Additional form of the MULTIPLY statement.
   (MULT)

The following expansions have been made to the DATAFORM 2 compiler to produce the DF11 compiler:

1. A variable, rather than a literal, is required for the CHAIN statement.
2. More label space (255 maximum) is provided.

The following expansions have been made to the DATAFORM 2 form generator to produce the DF11 form generator:

1. New edit criteria are available via the REQUIRED pass:
   - K - keyin-continuous
   - X - required and keyin-continuous
2. New edit criteria are available via the JUSTIFY pass:
   - J - right justify
   - Z - zero fill
   - R - right justify and zero fill
3. New edit criteria are available via the TYPE pass:
   - O - numeric format, minus-overpunch
   - S - alphabetic format, shift inversion

The following expansions have been made to the DATAFORM 2 interpreter to produce the DF11 interpreter:

1. Operator record delete capability (DISPLAY/5) has been added.
2. 16.8 rather than 12.4 characters of precision are available in DF11 numeric fields.
3. Initial data entry mode specification may be placed on the command line.

The DATAFORM 2 configurator has been eliminated, and has been replaced by a print utility for printing forms (with and without the assigned edit criteria) and data.

FORM GENERATION AND TEST
without programs

```
DF11GEN  ------> form  -------> DF11  --------> data
```

FORM GENERATION AND TEST
with programs

```
EDIT  -----> program  ------> DF11CMP  -------> program
             source                             object
                                                   |
                                                   |
                                                   |
                                                   |
  -------------------------------------------------|
  |
  |
  |
  |
  V                 form
DF11GEN  -------> with     -------> DF11  ------->    data
                  program
```

CHAPTER 2. GENERAL DATAFORM TERMS AND CONCEPTS

2.1 What is a FORM?

A "FORM" in this User's Guide refers to the processor's screen image. This screen image is created by the form generator. It contains labeling information, defines the length and positions of "data fields", and reserves space for "keyin only 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 may also be used to assign edit criteria to the data fields. The criteria are assigned field-by-field in separate passes over the form image.

These criteria include the field type:

- Alphabetic
- Alphanumeric
- Numeric digit
- Numeric formatted
- Numeric minus-overpunch
- Shift key inversion

justification:

- Left justified/blank filled
- Left justified/zero filled
- Right justified/blank filled
- Right justified/zero filled

entry restrictions:

- Fill controlled
- Key continuous
- Program reserved
- Required
- Required/fill controlled
- Required/key continuous
- Required/program reserved

semi-constant data; 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. The same field program may be assigned to more than one field.

Special function keys, which are discussed in the chapter on the form generator, enable cursor, character, line, and screen manipulation.

The screen image, basic edit criteria and field programs, if any, comprise the "form" which is subsequently interpreted by the DF11 interpreter.

2.2 What is a FIELD PROGRAM?

If extended editing and basic computation are required in a form, a program written and compiled in the DF11 language is necessary. This language provides access to the entire data record (on a character or field basis) and definition of working storage variables, tables, messages, etc. COMMON storage is available to pass information between forms. The DF11 language provides the following editing capabilities:

- Arithmetic
  - Add
  - Divide
  - Multiply
  - Subtract
- Data Manipulation
  - Align
  - Convert
  - Field number
  - Lookup
  - Move
  - Set
- Data Entry Control
  - Change
  - Entrymode
  - Modifymode
  - Reset

Data Checking
- In range
- In table
- Not in range
- Not in table
- Null
- Retry

Check Digits
- Ck10
- Ck11

Comparisons
- Equal
- Greater than
- Greater than or equal
- Less than
- Less than or equal
- Not equal

Branching
- Again
- Call
- Chain
- Go to
- Next
- Return
- Store

Output
- Backspace
- Beep
- Close
- Delete
- End
- Formshow
- Message
- Peof
- Read
- Show
- Weof
- Write

Data Definition
- Common
- Data
- Equ
- Field
- Redefine
- Reserve
- Work

Data Buffers
- Input
- Output

The field programs may be assigned to particular fields in a pass of the form generator. When the form is written out, the relocatable program will be converted to "absolute" code and written to the form file.

During data entry, 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.

2.3 User Space and How It´s Allocated

When a new form is being created, there are 5000 characters of memory, called "user space", available. This "space", however, encompasses all the following:

- Common storage
- Field programs (if required)
- Form image
- Keyin data buffer
- Writing data buffer

The form generator indicates the amount of free space as soon as the form image has been defined. The field programs and COMMON storage must fit in the remaining free space.

2.4 Some Data Entry Features

In conjunction with the DISPLAY key, the number pad keys can provide the operator with the following functions:

- Backspace field
- Backspace record
- Delete record
- Form data duplication
- Form data erase
- Load next form
- Return to Read record
- Monitor
- Write record

If semi-contant data is defined in the form, it may be accepted or overwritten by the data entry operator. Constant data cannot be overwritten, and is placed in the data record as is.

Forms may be loaded in any 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-writing in-place permitted.

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

CHAPTER 3. THE FORM GENERATOR

A DF11 "form" is an image displayed on the processor's screen which contains form text (explanatory information for the operator, not to be written to the data file), field definitions (special characters which define a field to be filled in by the operator and to be written on the data file) and keyin space (special characters which define a field to be entered [but not stored in the data record]). The processor's screen is 80 characters wide and 12 lines high and any of the 960 positions on the screen may be used in the form.

Each form is contained in a file named "SYSNAMnn/DFF", where "SYSNAM" is the name of a system of forms which may reference each other and "nn" is a two digit number assigned to a particular form. How to load the generator, the filenames required, and default conditions for filenames required is discussed in chapter one.

3.1 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 or non-underscore character 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 used for entering and one used for writing data), and each field generates a six character set of edit criteria. Each field defined has a "field number" corresponding to its relative position in the form (and pointing to its entry in the edit criteria 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.

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

1) Only one set of edit criteria applies to the 3-character field whereas each 1-character field may be assigned different sets of edit criteria.

2) Since each set of edit criteria takes 6 characters, the three 1-character fields use more user space than the single 3-character field.

3) Only one field program may be assigned to the 3-character field, whereas each 1-character field may have its own field program.

4) The single 3-character field may be right justified and/or zero filled.

Fields defined by carets will be "space compressed" in the form image (BUT NOT IN THE DATA RECORD!). 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. Space compressed fields allocate less "user space" than non-compressed fields.

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 a field definition</u>.

3.2 Keyin Only Field

A keyin only field, with the exception of the initial character, is defined exactly as is a data field. Keyin only fields begin with a less than character (<) and are continued by carets or underscores. They may appear anywhere in the form. Keyin only fields create a six character set of edit criteria like other fields and thus have a corresponding "field number". However, no space is reserved for these fields in the data record. A keyin only field may be used as a verify field, or as a program message field. Nothing in a keyin only field ever gets written to the data file.

3.3 User Space

There is a fixed amount of space available which must contain the form image, the data input/output areas, the edit criteria table, and field programs. This fixed area is called "user space". 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 (249) which can be defined. The total user space available is 5000 characters.

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 character set of edit criteria. The characters displayed in the form image, both labeling 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 criteria 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 criteria table and form image, user space may be allocated to field programs. The length of a field program is indicated on the listing and on the screen at the end of program compilation.

When the form is written to the form file, the amount of user space remaining (or the excess allocated, if any) is displayed on the screen. If an excess is allocated, either the form or (if present) the field programs should be revised.

3.4 Form Worksheet

To aid in the design of forms, a "DATAFORM Worksheet" is available. This worksheet provides space for designing the screen image and for recording the various edit 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.

3.5 The NEW Command

To generate a _new_ form, enter the:

```
NEW
```

command to clear the screen and enter the image generation mode.

Titles and field definitions may be entered. Pressing the ENTER key places the cursor at the beginning of the next lower line; pressing ENTER without entering text leaves a blank line in the form.

Additional form manipulation is available with the DISPLAY key and the keys on the number pad. When the DISPLAY key is pressed, the keys in the number pad to the right of the keyboard (or the regular number keys) become a set of _special function keys_ enabling: the movement of the cursor up, down, left and right; the insertion and deletion of characters; the deletion of words; the insertion of lines; and the erasure of lines and portions of the screen.

A key becomes a special function key if it is pressed simultaneously with the DISPLAY key. That is, holding down the DISPLAY key while pressing the desired number key turns the number key into a special function key.

The following is a summary of the special function keys:

7	Character insert
8	Cursor up
9	Erase to end of screen
4	Cursor left
5	Duplicate character
6	Cursor right
1	Word remove
2	Cursor down
3	Line insert
0	Remove character
.	Erase to end of line
CANCEL	Return to monitor

Additionally, the CANCEL key (_not the CANCEL function key_) will erase an entire line.

3.5.1 Repeat Key (KEYBOARD)

The KEYBOARD key causes a character (and many functions) to be repeated. That is, holding down the KEYBOARD key while pressing a character causes the character to be repeated as long as the KEYBOARD key is held down. Also, holding down the DISPLAY and KEYBOARD keys while pressing a number pad key causes the special function key to be repeated.

3.5.2 Cursor Movement Function Keys (2,4,6,8)

There are four cursor movement function keys which are non-destructive; i.e., they pass over characters on the screen without erasing them. The cursor down function key (2) moves the cursor DOWN, the cursor up function key (8) moves the cursor UP, the cursor right function key (6) moves the cursor RIGHT and the cursor left function key (4) moves the cursor 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 preceding line, except, of course, on the top 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 function keys may be repeated.

3.5.3 Character Insert Function Key (7)

The character insert function key (7) at the upper left of the number pad, opens a space for character insertion wherever the cursor is positioned on the screen. This function key may be repeated. Characters at the right most edge of the screen are truncated, not wrapped around.

3.5.4 Character Remove Function Key (0)

The character remove function key (0) at the lower left of the number pad, causes the character at the cursor to be removed and the remaining characters to be concatenated to the left. The line is blank filled on the right. This function key may not be repeated.

3.5.5 Erase Function Keys (1,.,9)

There are several keys available to erase all or part of the screen image. The erase function keys may not be repeated. The word remove function key (1) causes a word (that is, a group of characters edged by spaces) to be removed. The line is concatenated, and blank filled on the right. The cursor may be placed anywhere in the word when the word remove function key is pressed.

The erase to end of line function key (.) causes the line to be erased from the position of the cursor to the right hand edge of the screen.

The erase to end of screen function key (9) causes all characters to be erased from the cursor to the end of the screen, i.e., through line 12 character 80. This key could be used clear the entire screen, if the cursor were placed in the upper left corner of the screen.

The CANCEL key (not the CANCEL function key) causes the entire line that the cursor is on to be erased, and places the cursor in the first position of the line.

3.5.6 Line Insert Function Key (3)

The line insert function key (3) causes a blank line to be inserted _at_ the line where the cursor is blinking. The line at the cursor and all lower lines are _rolled_ _down_ the screen one line. The twelfth line will disappear. This function key may not be repeated.

3.5.7 Duplicate Character Function Key (5)

The duplicate character function key (5) 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 when the cursor is placed on the top line of the screen.

3.5.8 Return to Monitor Function Key (CANCEL)

When the screen has the desired appearance, the return to monitor function key (CANCEL) function key returns control to the generator's monitor. At this point the generator displays the message:

```
nnn DATA

mmm BYTES LEFT
```

indicating the number of characters in the data record and the number of characters remaining in the user space. If the number of characters in the data record is greater than 249, the generator displays the message:

```
MORE THAN 249 DATA
```

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

```
MORE THAN 126 FIELDS
```

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

If the combined space required by the form image, data areas and sets of edit criteria exceeds the available user space, the generator displays the message:

```
nnn BYTES OVER
```

The form should be revised to fit the user space available. Suggestions on saving space are discussed in an APPENDIX.

3.6 Assignment of Edit Criteria

When the form image has been generated, the form is still only in memory and no edit criteria have been assigned.

Edit criteria may be assigned to each field of a form. Different kinds of edit criteria may be assigned in different "passes" over the fields of a form. Each type of edit-defining pass (TYPE, REQUIRED, JUSTIFY, SEMI-CONSTANT, CONSTANT, PROGRAM, LINK) must be requested separately, and, finally, the form must be written to the form file 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.

During each pass, the form is redisplayed with the cursor at the first field definition (i. e., the first vertical bar (|) or less than (<) sign). Any one of the accepted edit criteria for that pass may be assigned, the field may be bypassed without changing or assigning the edit criteria (by pressing the ENTER key), or the edit criteria may be cleared (by pressing the CANCEL key).

If a pass is re-executed, the current edit criteria will be displayed as each field is reached. If no change is needed, pressing the ENTER key proceeds from field to field.

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

To request a pass, enter the name of the pass. Only the first 3 letters of the pass need to be entered to initiate the pass.

3.6.1 The TYPE Pass

The TYPE pass is entered to set restrictions on the characters which may be entered into a field. The acceptable types for this pass are discussed below.

If no TYPE edit criteria is assigned to a field, any character is acceptable in any position of that field.

3.6.1.1 Alphabetic (A)

The alphabetic edit criteria for the TYPE pass (A) indicates that characters entered must be uppercase alphabetics (A through Z) or space.

3.6.1.2 Digit (D)

The digit edit criteria for the TYPE pass (D) indicates that characters entered must be strictly numeric (0-9).

3.6.1.3 Numeric (N)

The numeric edit criteria for the TYPE pass (N) indicates that characters entered must be of the set of: digits (0-9), a decimal point, or a minus sign (plus signs are not allowed).

During data entry, numeric fields are 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 sixteen positions are permitted to the left and eight to the right of the decimal point.

3.6.1.4 Mixed (M)

The mixed edit criteria for the TYPE pass (M) indicates that characters entered must be of the set of: Alphabetics, space, digits, decimal point, or minus sign. No other special characters are allowed.

3.6.1.5 Numeric Minus-overpunch (O)

The numeric minus-overpunch edit criteria for the TYPE pass (O) indicates that the characters entered must be in numeric format. The exception is that the right most character (not the left) may be a minus sign. A minus sign in the rightmost position causes the character to the left of the minus sign to be "overpunched" with the minus sign. That is, an operator entered "0-" becomes "}"; a "1-" becomes a "J"; a "2-" becomes "K"; etc.

If the field is assigned the "zero fill" edit criteria in the "JUSTIFY" pass the overpunch will occur in the rightmost position.

Minus overpunch fields should not be assigned the "fill-control" or "key-continuous" edit criteria in the REQUIRE pass.

3.6.1.6 Shift Key Inversion (S)

The shift key inversion edit criteria for the TYPE pass (S) indicates that the alphabetic characters entered are not to be converted to capital letters unless the shift key is depressed. That is, the shift key has the same effect as it does on a standard typewriter keyboard.

3.6.2 The REQUIRE Pass

The REQUIRE pass is entered to establish that a field may not be bypassed (tabbed past without entering data) during data entry, or that all characters must be entered, or that the field is not to be filled by an operator but is to be filled by a field program.

If no REQUIRE edit criteria is assigned to a field, the ENTER key must be pressed somewhere in the field to proceed to the next field.

3.6.2.1 Required (R)

The required edit criteria for the REQUIRE pass (R) indicates that a field is required. This means that during data entry, at least one character must be entered into the field.

3.6.2.2 Fill Controlled (F)

The fill controlled edit criteria for the REQUIRE pass (F) indicates that a field is to be fill controlled. This means that during data entry, the field must be completely filled by the operator.

Fields whose edit criteria for the JUSTIFY pass is J, Z, or R should not be fill controlled. For these fields, the interpreter aligns the data after the ENTER key is pressed.

Fill control fields may be bypassed, however, if the ENTER key is pressed in the first column of the field. The ENTER key is an unacceptable key elsewhere in the field.

3.6.2.3 Required and Fill Controlled (B)

The required and fill controlled edit criteria for the REQUIRE pass (B) indicates that a field is both required (R) and fill controlled (F). The ENTER key is an unacceptable key.

3.6.2.4 Program Reserved (P)

The program reserved edit criteria for the REQUIRE pass (P) indicates that a field will be filled by a field program. No operator keyin is permitted in this field.

This edit criteria may also be set on a keyin only field to reserve it as an alternate message display area.

3.6.2.5 Required and Program Reserved (S)

The required and program reserved edit criteria for the REQUIRE pass (S) indicates that a field is to be both program reserved (P) and required (R). This will prevent writing of the data record if data has not been entered into the program reserved field by a field program.

3.6.2.6 Keyin Continuous (K)

The keyin continuous edit criteria for the REQUIRE pass (K) indicates that a field may be terminated either by pressing the ENTER key or by entering the last character (as in fill controlled fields).

3.6.2.7 Required and Keyin Continuous (X)

The required and keyin continuous edit criteria for the REQUIRE pass (X) indicates that a field is both required and keyin continuous.

3.6.3 The JUSTIFY Pass

The JUSTIFY pass is entered to either right justify (rather that the default left justify) or zero fill (rather than the default blank fill) a field.

3.6.3.1 Right Justify (J)

The right justify edit criteria for the JUSTIFY pass (J) indicates that a field is to be right justified and blank filled to the left.

3.6.3.2 Zero Fill (Z)

The zero fill edit criteria for the JUSTIFY pass (Z) indicates that a field is to be zero filled on the right.

3.6.3.3 Right Justify and Zero Fill (R)

The right justify and zero fill edit criteria for the JUSTIFY pass (R) indicates that a field is to be right justified and zero filled on the left.

3.6.4 The SEMI-CONSTANT and CONSTANT Passes

The SEMI-CONSTANT or CONSTANT pass is entered to set semi-constants or constants into a field in a form. Semi-constants and constants are characters set into a data field in the form image. During data entry the operator has the option to <u>accept or over-write data set by the SEMI-CONSTANT pass</u>; whereas, data set by the <u>CONSTANT</u> pass automatically becomes part of the data record and <u>cannot be rejected</u> by the operator. Both commands cause the form to be displayed with the cursor in the first field capable of accepting constant or semi-constant information.

Semi-constants and constants may only be set in fields initially defined at image generation time by underscores.

In the CONSTANT pass, the SPACE bar does not set constant spaces into the field but permits movement to the 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 (_), vertical bars (|) or carets (^) can be set within the field. The CANCEL key will clear any constant field previously set. The BACKSPACE key positions back one character and erases the last character entered.

During the CONSTANT pass, no editing is performed on constants entered. Unaceptable constants will cause the interpreter to <u>hang beeping</u> during data entry. Unacceptable semi-constants will be displayed. This feature may be useful for presenting prompting information to the operator, e.g., a date field may have the unacceptable semi-constant "YYMMDD" set to guide the operator.

Also, an entire form of constant data should not be prepared; at least one position must be left for the operator - so that the form may be viewed and/or written to the data file. All-constant forms (or forms with no fields) will cause the interpreter to hang clicking 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 enter data over them to enter the remainder of the field.

Once semi-constants or constants have been set, they will always appear when the form is displayed (e.g., during the TYPE or REQUIRE pass). Semi-constants and constants are not destroyed by assigning edit criteria during other passes.

Semi-constants and constants should be cleared before executing the REVISE command since their presence will change the field definitions.

3.6.5 The PROGRAM Pass

The PROGRAM pass is entered to assign field program names to fields. Field programs are written in the DF11 language, which is discussed in a later chapter. Each program is identified by a single alphabetic character (A - Z). A program is assigned to a field by entering the appropriate program letter in any field where a special processing program will be written.

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.

3.6.6 The LINK Pass

The LINK pass is entered to assign a "link" to another form so that the operator need never be concerned with a form number. Each form in a DF11 system may have a pointer, called a "link", to the next form to be used. This pointer must be defined at form generation time. Form links should be planned carefully so that forms are accessed in a manner most convenient to the operator.

NOTE: LINKed forms must have the same SYSNAM.

A form link may be either of two types: a manual link or an automatic link. The operator must press a special function key to load a manual linked form after the data record has been written. An auto linked form is automatically loaded whenever a data record is written.

When the LINK pass is entered, the message:

```
NEXT FORM nnn:
```

will appear (where nnn is the number of the current linked form in octal, initially 000). The current linkage information may be viewed by entering the LINK pass and then simply pressing the ENTER key to leave the value unchanged.

3.6.6.1 Setting a Manual Link

To set a manual link, enter the number of the form (followed by the ENTER key) which is to be displayed when the operator presses the form load function key.

3.6.6.2 Setting an Auto Link

One data entry transaction may require several DATAFORM "forms", e.g. forms 1, 2 and 3 (PAY01, PAY02 and PAY03) may 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).

To facilitate use of multiple 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 a new form whenever data is written.

To set an auto-link precede the form number with a minus sign. Thus, when generating form one in the multi-page example above, enter "-2" as the auto link for form 1; enter "-3" as the auto link for form two; and "-1" as the auto link for form 3 (which makes form three wrap around to form one).

3.6.6.3 Clearing a Link

To clear a form link, enter a zero when the "NEXT FORM" message is displayed.

3.7 The OUT Command

<u>During the entire form generation time the form is only in memory</u>. To record the form and its associated edit criteria in the form file, enter the:

```
OUT
```

command. If no errors have been detected (e.g. too many fields, too long a data record), the form will be written. If programs have been specified, the program file (see chapter 1 for a discussion of where the program file name originates) will be opened and searched for all referenced programs. If the file or any of the programs are missing, an error message is displayed and the form is written without field programs.

At the completion of the form writing process, the generator displays either the message:

```
PROGRAM BASE ADDRESS mmmm

nnn BYTES LEFT
```

and reloads the DOS or the message:

```
nnn BYTES OVER
```

This message means that the form image plus the data record plus the field program is too large to be contained in available user space. Either the form or the field programs must be revised to fit into the user space. All numbers including the address displayed here are decimal.

When the new form has been written, it may be tested by running DF11 specifying the newly created form.

3.8 The REVISE Command

If an error in the form image is discovered after the image has been generated, the:

```
REVISE
```

command places the generator in the image generation mode with the current form intact. <u>All edit criteria are cleared</u> which means that <u>all passes have to be re-executed</u> after the form has been revised.

If the form is not in memory, the OLD command must be entered before the REVISE command to load the old form into memory.

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

3.9 The OLD Command

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

    OLD

command loads the form into memory. Any pass of the generator may be executed; however, note that the REVISE command will clear all edit criteria.

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

3.10 The OS Command

The:

    OS

command reloads the DOS without writing the current form in memory to the form file.

GENERATING A NEW FORM

```
                           NEW
                     make form image

 TYPE      REQ       JUS       SEMI      CON        PRO       LINK
assign    assign    assign    define    define     assign    set
edit      edit      edit      semi-     constants  program   manual
criteria  criteria  criteria  constant             letters   or auto
                                                              link

                            |
                            |
                            |
                            |
                            |
                            |
                           OUT
                  write form to form file
```

CHAPTER 4. THE COMPILER

The DF11 interpreter provides field editing capabilities on a character-for-character basis. Field programs written in the DF11 language provide much greater field editing capabilities. The DF11 language is a high level programming language, similar in structure to DATABUS and other high level languages. A field program can perform almost any kind of field (and even character) manipulation: check digit, range, and table checks; complete arithmetic processing; inter-form communication; complex data record movement; code-set conversions; etc.

The DF11 language is concise, yet powerful. The basic ingredients of the language are, as in any programming language, statements which describe data (called "specification" statements in the DF11 language), and statements which manipulate data (called "executable" statements).

4.1 Labels

Any DF11 statement may have a label, and some _must_ have a label. A "label" begins in column one and consists of up to eight alphanumeric characters (actually, the label may consist of any number of alphanumeric characters, although all characters after the first eight are ignored).

Labels have three uses: first, to name data items; second, to provide a means for branching and subroutine calls within a DATAFORM program; and third, to name field programs (that is, to associate program code segments to specific fields in the form image).

At most 246 labels may be defined in a DF11 compilation.

The following are examples of acceptable labels:

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

4.2 Field Program Names

The form generator uses a label called a "field program name" to associate a specific starting address of a DF11 program segment with a specific field of a form. A field program name is a label which is terminated by a star (or asterisk) "*", and there are no blanks between the label and the star. Since only the first character of a field program name is passed to the form generator, it is pointless (and probably could be confusing) to name field programs with labels which are longer than one character. In addition, the generator requires an alphabetic field program name. It is important to note that the compiler does not check for duplicate field program names; if there are duplicates, it passes both to the generator.

The following are examples of program names:

```
E*
Z*
```

4.3 Spaces

The DF11 compiler is a "free-form" compiler -- that is, the space character ( ) is by and large ignored by the compiler. Multiple spaces are treated as a single space, and a single space is ignored except as a field separator. Spaces may be included as desired to improve readability.

4.4 Comments

Comments, too, are ignored by the DF11 compiler.

There are two kinds of comments -- comments which appear on a code line after the code; and comments which appear on a line by themselves. Comment lines must begin with a period (.) or a plus (+) in column 1. If a listing is printed, a comment that begins with a plus causes a page to be ejected on the printer and the comment line to be printed on the top line of the next page of the listing.

4.5 Specification Statements

As mentioned earlier, specification statements are statements which describe data. The DF11 language contains: the DATA statement (used to access the output data record); the WORK statement (used for data storage <u>within</u> a single form); the RESERVE statement (used to change the size of COMMON); the COMMON statement (used for data communication between forms); the EQU statement (used to describe absolute values); the REDEFINE statement (used to associate a label with a previously defined label); and the FIELD statement (used to describe fields of the screen image form).

Every specification statement has associated with it an "item length". The item length is the number of characters which make up an individual item of that statement. The item length of each specification statement below is the length of the entire statement, unless otherwise indicated.

4.5.1 DATA

The DATA statement refers to specific columns of the OUTPUT data record. The general format of the DATA statement is:

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

where "n" and "m" are decimal numbers in the range 1-249. 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 item length associated with the DATA statement is: (m-n)+1. The columns defined by the DATA statement do not necessarily correspond to specific fields of the form. Areas may be redefined. The columns defined by a 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.

The following syntax restrictions apply to the DATA statement:

1) "n" and "m" must both be greater than zero but less than 250.
2) "m" must be greater than or equal to "n".
3) The DATA statement must have a label.

Examples of the DATA statement:

```
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
```

4.5.2 WORK

The WORK statement is used to reserve space within a field program. Space reserved may be uninitialized, or may contain ASCII or octal constants (or tables).

To simply reserve uninitialized space within a field program, the following format of the WORK statement is used:

```
          <label>     WORK <n>
```

where <n> is a decimal number in the range 1-249. The area to which <label> refers has an item length of <n>.

Working storage may contain ASCII characters. The characters are enclosed in double quotation marks, as in the following example:

```
          WORDS       WORK "PRE-DEFINED CHARACTERS"
```

A special forcing character, (#) may be 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:

```
          NICKNAME    WORK "I AM #"SHORTY#"."
          NUMBER1     WORK "I AM ## 1."
```

Each WORK statement that contains constants generates a code segment. Normally, every constant working storage segment is terminated with an additional, special end-of-table character, an octal zero. This character is included in the over-all length of the working storage segment, but is not included in the item length. To conserve memory, it is possible to suppress the special end-of-table character in a constant working storage segment by following the last item of the working storage segment with a semicolon, as in the following examples:

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

The first example will generate the following five octal characters: 0104,0101,0124,0101,000. The second will generate the following four octal characters: 0104,0101,0124,0101. The item length of both statements above is four.

Working storage may contain tables as well. The item length of the table is determined by the length of the first item in double quotation marks. Each item in the WORK statement table must be the same length. Individual items are separated by a comma.

In the following examples:

```
          TABLE1      WORK "1","2","3","4","5","6"
          TABLE2      WORK "12","34","56"
          TABLE3      WORK "123","456"
          TABLE4      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.

Working storage items may be continued on more than one line by using a colon, as in this example:

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

Working storage may contain octal constants. The first octal constant (and only the first) is prefixed by the alphabetic letter "O". Each octal constant generates only one character of working storage. An octal constant may consist of any number of octal digits; however, only the least significant eight bits are placed in the octal character. Octal constants may be separated from one another by a comma, and may be continued from one line to another by use of the colon. Octal constants, like other constants, are terminated with an octal zero; a semicolon after the last constant will suppress the zero. The item length of an octal constant work 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.

The following are examples of octal WORK statements:

```
          OCTAL1      WORK O15;
          OCTAL2      WORK O15,16,17,20
          OCTAL3      WORK O15,16,17,20:
                           25,26,27,30:
                           35
          OCTAL4      WORK O107
```

The following syntax restrictions apply to the WORK statement:

1) The WORK statement must have a label.
2) If the WORK statement defines a table, all items in the table must be of the same length.
3) A comment may appear on a WORK statement if the comment is preceeded by a period.
4) If the WORK statement merely reserves space (i. e., does not contain any constants), the amount of space reserved must be in the range 1-249.

4.5.3 COMMON

The COMMON statement is used to assign labels and reserve space within the COMMON block. 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. The format of the COMMON statement is:

```
[label]     COMMON <n>
```

The following example could be used to pass a six character total from one form to another:

```
TOTAL       COMMON 6
```

It is important for every program using information saved through COMMON to have the same relative locations of areas inside the COMMON block. References to COMMON data in second and subsequent form's programs must be in the same order. A dummy COMMON statement, such as:

```
DUMMY       COMMON 6
```

should be used to skip over 6 unused characters inside the COMMON block, if those characters are not referenced by the current form, but are referenced by another form.

The following syntax restrictions apply to the COMMON statement:

1) A label is not required on a COMMON statement.
2) The maximum total length of the COMMON block is 100 characters unless the length is changed by the RESERVE statement.
3) A comment may appear on a COMMON statement if the comment is preceeded by a period.

4.5.4 RESERVE

The RESERVE statement is used to change the size of the COMMON block. The COMMON block is initially set to 100 characters. The format of the RESERVE statements is:

```
RESERVE <n>
```

where <n> is the size of the COMMON block to be reserved. The RESERVE statements must appear before all COMMON statements.

NOTE: all forms in a system should have the same size COMMON block, to prevent any destruction of COMMON data.

4.5.5 EQU

The EQU statement is used to associate an octal address value with a label. 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.

Previously assembled assembly (as distinct from DF11) language programs may be referenced by using the EQU statement to define a label, and then transferring control to that label (see later sections of this manual for transfer of control statements and for assembly language interfacing).

The following are examples of the EQU statement:

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

4.5.6 REDEFINE

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

The general format of the REDEFINE statement is:

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

The value "n-1" is added to the previously defined initial value for <label1> and becomes the initial value 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 statement may redefine WORK and COMMON statements (and the pre-defined label INPUT).

The following syntax restrictions apply to the REDEFINE statement:

1) Both <n> and <m> must be in the range 1-249.
2) The REDEFINE statement must have a label.
3) The field following <m> may be used as a comment field.
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 statement does not immediately follow the label that is being redefined.

4.5.7 FIELD

The FIELD statement is used to reference the OUTPUT fields of the displayed form. The field reference may be absolute or relative to the current field. The absolute field reference is used to reference _specific_ fields of the form.

The format of the absolute FIELD statement is:

```
<label>     FIELD <n>
```

where "n" is a decimal number in the range 1-126.

The relative field reference is used to reference an offset (either positive or negative) of the _current_ field.

The format of the relative field statement is:

```
<label>     FIELD <sign><n>
```

where <sign> is either a "+" or a "-", and "n" is a decimal number in the range 1-126.

The following are examples of the FIELD statement:

```
FIELD7      FIELD 7
NEXTFLD     FIELD +1
LASTFLD     FIELD -1
```

The label appearing on a FIELD statement may be referenced in any type of arithmetic or conditional statement, as in the following example:

```
            ADD LASTFLD TO INPUT GIVING NEXTFLD
```

4.6 Executable Statements

Executable statements are those statements concerning: 1) transfers of information; 2) arithmetic; 3) comparisons; 4) output; 5) transfers of control; and 6) current field assignment.

4.6.1 Transfers of Information

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

4.6.1.1 ALIGN

The ALIGN statement 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.

In the following example, the source field and the destination field (both before and after the ALIGN) are shown:

```
          MOVEIT      ALIGN FIELD1 TO FIELD2
```

FIELD1	FIELD2	FIELD2	
		(before)	(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
	12.34	00000	00012

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.

4.6.1.2 CONVERT

The CONVERT statement format is:

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

The CONVERT statement 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>.

Given the following specification statements:

```
          TABLE1       WORK "MA","NY","KS","MT","TX"
          TABLE2       WORK "BOSTON","ALBANY","TOPEKA":
                            "HELENA","AUSTIN"
```

and the following executable statement:

```
          CONVERT FIELD1 BY TABLE1 AND TABLE2 GIVING FIELD2
```

the following will be the contents of FIELD2 if the contents of FIELD1 are as indicated:

FIELD1	FIELD2
TX	AUSTIN
MA	BOSTON
KS	TOPEKA

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.

The CONVERT statement should be used when the table has gaps, or is randomly ordered.

4.6.1.3 FIELDNO

The FIELDNO statements places the currect field number (in ASCII) in the area specified by <label2>. The format for the FIELDNO statement is:

```
          [label1] FIELDNO <label2>
```

4.6.1.4 LOOKUP

The LOOKUP statement format is:

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

The LOOKUP statement 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 following is an example of the LOOKUP statement:

```
        TABLE       WORK "JAN","FEB","MAR","APR","MAY","JUN":
                         "JUL","AUG","SEP","OCT","NOV","DEC"
                    LOOKUP NUMBER IN TABLE GIVING NAME
```

The LOOKUP statement should be used when there are no "gaps" in the table from which the data movement takes place. The LOOKUP uses <field1> as an item by item index into the table, and hence will always find a match, even though it may be outside the range of the table (if the index is too large).

4.6.1.5 MOVE

The MOVE statement 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, meaning that some characters may be truncated (or lost). An example of the move statement is:

```
                    MOVE TOTAL TO WORK1
```

4.6.1.6 SET

The SET statement 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 following example:

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

would set the entire 8 character TOTAL field to stars. The SET 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 the decimal, too, will be overstored.

4.6.2 Add, Subtract, Multiply, Divide

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 MULT
                      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, quotient) appears at the third label rather than the second.

NOTE: A comment may appear on an arithmatic statement if the comment is preceded by a period.

The following are examples of arithmetic statements:

```
                      ADD INPUT TO SUBTOTAL
                      SUB DISCOUNT FROM PURCHASE
                      MULTIPLY PRICE BY QUANTITY
                      DIVIDE TOTEST INTO TOTSCORE
                      ADD INPUT TO OLDBAL GIVING NEWBAL
                      DIV TOTEST INTO SCORE GIVING AVESCORE
```

If GIVING <label3> is appended to an arithmetic statement, an "ALIGN <label2> TO <label3>" is generated prior to the arithmetic statement.

NOTE: Significance may be lost with GIVING <label3> (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 zeros are supplied in non-significant leading characters. In a field defined as right justified and blank filled, performing an "ADD NULL TO <field>" will replace the leading blanks by zeros.

NOTE: Arithmetic should not be performed on minus overpunch fields. The result of any arithmetic using minus overpunch fields is indeterminate.

4.6.3 The IF Statement

The general format of the IF statement is:

```
[label1]   IF <field1><relation><field2> THEN <label2>
```

If <relation> is true, control is transfered to <label2>, which may be a pre-defined label like STORE. If <relation> is false, the next statement in the program is executed. 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 characters in <field1> are compared, from left to right, to the characters in <field2> (using the item length of field1 to terminate the compare). Differing lengths do not cause unequal compares; however, if <field1> is longer than <field2>, the results are indeterminate. Comparisons of minus overpunch fields are indeterminate.

2) Table lookup (INR, INRANGE, INT, INTABLE, NIR, NOTINRANGE, NIT, NOTINTABLE). <field1> is "looked-up" in the table defined at <field2>. The item 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>.

The following are examples of the usage of the IF statements:

```
AMOUNT       FIELD 1
ACCOUNTNO    DATA 21,27
MONTH        DATA 1,2
DAY          DATA 3,4
DAYTABLE     WORK "01","31"
MONTHTABLE   WORK "01","12"
ZERO         WORK "000000"
WEIGHT1      WORK "212121"
.
. Check field1 for strictly positive
.
A*           IF AMOUNT GREATER ZERO THEN STORE
             AGAIN
.
. Check for null input
.
B*           IF NULL EQ INPUT THEN AGAIN
.
. Check for negative.
.
C*           IF AMOUNT 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
```

4.6.4 Output Control

The BEEP statement provides an audible tone. The CHAIN statement is used to load another form (in addition to the auto-load and linking-load features of the interpreter). Three statements are provided for displaying information on the processor's screen: FORMSHOW, MESSAGE, and SHOW. The WRITE, READ, WEOF, BACKSPACE, DELETE, and PEOF statements are used to perform functions similar to the function keys and commands available to the operator.

4.6.4.1 BEEP

When the BEEP statement is executed, the processor issues a single BEEP sound. The format of the BEEP statement is:

```
          [label]     BEEP
```

4.6.4.2 CHAIN

The CHAIN statement loads a specific form. The format of the CHAIN statement is:

```
          [label]     CHAIN <label2>
```

where <label2> is a work area which contains the decimal number of the form to be loaded (from 01 to 99). A file named "SYSNAMnn/DFF" is loaded. The current data record is not written; however, the flag indicating data present is cleared. The form is loaded and control is passed to the interpreter at the first non-constant field of the new form.

A CHAIN to the form currently in memory reloads that form and all its programs.

4.6.4.3 FORMSHOW

The FORMSHOW statement causes the current form to be redisplayed. All data fields on the screen will be cleared. The output record is not affected and the current field index is not changed.

The format of the FORMSHOW statement is:

```
          [label]     FORMSHOW
```

In the following example:

```
                      WRITE
                      FORMSHOW
```

the last data record written is still in memory; however, it will be erased from the screen and will appear only as each field is reached by the operator.

NOTE: The INPUT field is destroyed when the FORMSHOW statement is executed.

4.6.4.4 MESSAGE

The MESSAGE statement writes the specified message on the bottom line of the screen.

The format of the MESSAGE statement is:

```
            [label1]    MESSAGE <label2>
```

The following is an example of the MESSAGE statement:

```
            ERR         WORK "ACCOUNT IS OVERDRAWN"
                        MESSAGE ERR
```

The MESSAGE statement always erases the bottom line of the form. 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 when the MESSAGE statement is executed.

4.6.4.5 SHOW

The SHOW statement displays a message in the current field area of the screen.

The format of the SHOW statement is:

```
            [label1]    SHOW [label2]
```

If no [label2] is indicated, the SHOW statement defaults to the contents of the OUTPUT buffer corresponding to the current field.

The following are examples of the SHOW statement:

```
                        SHOW
or                      SHOW TOTAL
```

The SHOW may be used if computations or table lookup conversions were made to change the value of the current field, as in the following example:

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

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

NOTE: The INPUT field is destroyed when the SHOW statement is executed.

4.6.4.6 WRITE

The WRITE statement writes the data record to the data file. The format for the WRITE statement is:

```
           [label]     WRITE
```

Control is returned to the next statement in the field program. The data area in memory is not cleared, and may be used for further computation or for auto-duping selected data.

4.6.4.7 WEOF

The WEOF statement writes an end of file mark on the data file. The format for the WEOF statement is:

```
           [label]     WEOF
```

The interpreter read pointers are set to the end of file mark. The WEOF statement places the data file in MODIFYMODE.

4.6.4.8 READ

The READ statement reads the next record created by the current form into the data area. The format for the READ statement is:

```
            [label]      READ
```

The data file should be in MODIFYMODE for the READ and any subsequent WRITE statements to be executed properly. If an end of file mark is read, the data record will contain binary zeros.

4.6.4.9 BACKSPACE

The BACKSPACE statement backspaces the data file. The format for the BACKSPACE statement is:

```
            [label1]     BACKSPACE <label2>
```

where <label2> is a WORK area which contains the number of records to backspace. The number is a count of records created by the current form. The data file should be in MODIFYMODE for the BACKSPACE to be executed properly.

4.6.4.10 DELETE

The DELETE statement deletes the current data record. The format for the DELETE statement is:

```
            [label] DELETE
```

The entire record is over-written with delete characters (032), and written to disk.

4.6.4.11 PEOF

The PEOF statement positions the data file to the end of file mark. The format of the PEOF statement is:

```
            [label]      PEOF
```

File pointers are set so that the next WRITE operation will overwrite the end of file mark.

4.6.5 Transfers of Control

The three transfer of program control statements are the GOTO statement, the CALL statement, and the RETURN statement.

4.6.5.1 GOTO

Control is immediately transferred to the label following the GOTO:

```
                        GOTO <label1>
```

For the pre-defined labels, the word GOTO is optional. For programmer defined labels, it is mandatory.

The following are examples of the GOTO statement:

```
                        GOTO OVERDRAW
                        GOTO NEXT
                        NEXT
```

4.6.5.2 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>
                        RETURN
```

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

4.6.6 CHANGE and RESET

The CHANGE statement is used to transfer 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 from the current field number is specified immediately after the CHANGE statement:

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

For example, after the statement:

```
CHANGE +1
```

is executed, INPUT still contains the entered data; however, the current field number has been incremented by one and OUTPUT now reflects the position in the data record corresponding to the new field. After the statement:

```
CHANGE 1
```

is executed, however, the current field number has been changed to the first field in the form, that is, field 1.

When a field program is entered the number of the current field is saved and may be restored at any time. The:

```
[label]     RESET
```

statement will reset the field pointer to the field current when the program was entered.

4.6.7 MODIFYMODE and ENTRYMODE

The MODIFYMODE and ENTRYMODE statements allow the field program to control the data entry mode. The formats of these statements are:

```
[label]     MODIFYMODE
[label]     ENTRYMODE
```

The "mode" statements place the interpreter in the indicated mode. The data file is not affected in any way.

NOTE: The statements PEOF and ENTRYMODE when executed in that order duplicate the monitor's "ADD" command.

4.7 Pre-defined Labels

The nine labels discussed in this section may not be defined in DF11 programs. They have specific meaning to the DF11 interpreter, and are included automatically in every DF11 compilation.

The pre-defined labels INPUT, NULL, OUTPUT, and RETRY refer to locations within the interpreter. These four labels may be used as source or destination operands in data movement and comparison statements. Examples of the use of these labels are given below:

```
MOVE INPUT TO OUTPUT
```

```
                        IF NULL EQ INPUT THEN AGAIN
                        IF RETRY EQ NULL THEN STORE
```

The pre-defined labels AGAIN, CLOSE, END, NEXT and STORE cause a transfer of control from the field program back to the DF11 interpreter. These five 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
          D*            NEXT
          E*            CLOSE
          F*            END
```

AGAIN, CLOSE, END, NEXT and STORE are means of exiting a field program. It is important to note that the interpreter does not place data in the OUTPUT buffer before a field program is called. It is the responsibility of the field program to 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.

4.7.1 AGAIN

This label returns control to the interpreter at a point which indicates an error to the operator and re-requests the current field. That is, the processor BEEPS and returns the cursor to the first position of the field.

4.7.2 CLOSE

This label returns control to the interpreter at a point which closes the data file; displays the message:

```
          PROGRAM WRITTEN EOF
```

and reloads the DOS.

4.7.3 END

This label returns control to the interpreter at the point as if the operator had pressed the write data function key.

4.7.4 INPUT

This label designates the contents of the keyin buffer immediately prior to entering the field program. The data in INPUT has not yet been stored in the OUTPUT buffer. It's length is 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.

4.7.5 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.

4.7.6 NULL

This label designates a location in the interpreter which contains a binary zero. It may be used to determine if the data file is in normal data entry mode or modify mode; or if data is present in the OUTPUT record (meaning that this field had been entered before). The item length of NULL is always less than the item length of any variable. Therefore, in comparisons, NULL should be referenced first since the length of the first operand is used for the comparison.

4.7.7 OUTPUT

This label designates the contents of the data OUTPUT buffer 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 only fields.

4.7.8 RETRY

This label designates a location in the interpreter which contains a binary flag indicating whether the data file is in modify or data entry mode. RETRY can be checked by a field program by comparing RETRY to NULL. If RETRY equals NULL the data file is in data entry mode.

4.7.9 STORE

This label returns control to the interpreter at the point where the current contents of INPUT is transferred to the OUTPUT buffer. That is, exiting a field program through STORE is equivalent to:

```
MOVE INPUT TO OUTPUT
NEXT
```

4.8 Program Generation

Compilation of a program consists of two processes: using the DOS editor to create a new source program, or edit an existing program; and using the DF11 compiler to compile a new, newly edited, or old program.

4.8.1 Editing a Source Program

The commands of the general purpose editor are discussed in the chapter on EDIT in the DOS User's Guide. The name of the program file to be edited/created is indicated on the initial command line:

```
EDIT <program>;D
```

where ";D" indicates DF11 tabstops. Field program source file names should be in the "SYSNAMnn" format. The DF11 compiler and form generator all use the "SYSNAMnn" convention, and distinguish among files by their extensions.

4.8.2 Compiling a Source Program

When the source program has been edited, it should be compiled. This is accomplished by entering:

```
DF11CMP <sourcefile>[,objectfile][;options]
```

The compiler displays a sign-on message:

```
DF11 COMPILER 2.n - ddmmmyy
```

The compiler makes a first pass through the source file preparing a symbol table. The actual code generation and listing production take place on the second pass over the input file.

At the completion of the compilation, some or all of these messages are displayed on the screen:

```
    STORAGE USED IN DECIMAL: 00000 RELOCATABLE, 00000 COMMON
    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, 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 the DOS is reloaded.

Any error messages are automatically displayed on the screen, with a star indicating the part of the source line in error. The display may be stopped momentarily by pressing of either the KEYBOARD or DISPLAY keys.

4.8.3 Printing a Compilation Listing

The first action 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 three parts:

1) The line number.
2) The initial address (either absolute or relocatable) associated with the statement line.
3) The line as it was input.

If a listing is to be printed, the message:

```
          CODE TOO?
```

is displayed. A response of "Y" to this message will place the code generated for each line (eight characters per printed line, using as many lines as necessary for the amount of code generated) on the listing.

These listing options may be specified in the [;options] field of the DF11CMP command line. A semicolon (;) alone indicates that no listing is to be printed; a semicolon followed by: an "L" indicates that a listing is to be printed; a "P" indicates that the printer records are to be placed in a disk printer-image file (whose name is <sourcefile>/PRT) instead of on the printer; and a "C" together with either the "L" or the "P" indicates that generated code is to be included on the listing.

If a listing or print file is requested, a heading line may be entered.

4.8.4 The Program File

When compilation is complete, a file of the name "<sourcefile>/DFP" has been generated which contains the compiled code. The compiled code file consists of a header record and both relocatable and absolute object code records.

The header record contains the length of the relocatable object code, and the names and starting addresses of all field programs in the file.

4.9 Program Execution

4.9.1 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 entered, edited, and accepted by the interpreter. Thus, alpha-numeric checks, right justification, etc., will already have been performed on the input.

4.9.2 Operator Tabbing

If the operator chooses to bypass a field which is not required, INPUT is NULL (binary zero).

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 NULL) 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, INPUT is

NULL.

4.9.3 Pre-process Execution

To execute a field program as a "pre-process", the pre-process program should be assigned to a preceding field.

4.9.4 Program Reserved Fields

If a field is designated as a "program reserved" field, data for that field is to be assigned by a field program. When the field is entered, the field program is executed immediately and the area designated by INPUT is undefined.

4.9.5 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 entered. They will be edited and passed to the field program in the INPUT area. Unnaceptable constants will cause the interpreter to hang BEEPing during data entry.

CHAPTER 5. THE INTERPRETER

Data entry using DF11 involves loading the interpreter, then loading a form, selecting a data entry mode, and finally entering data into the fields defined by the form. When the data has been entered on the screen to the operator's satisfaction, and the data record has been written to the data file (by an operator function key or a field program instruction) then the same form is cleared and redisplayed with only constant and semi-constant data appearing.

The format for the DF11 command line is:

```
DF11 <SYSNAM[nn]>[,datafile][;options]
```

where SYSNAM is the name of the system of forms. The default form number value [nn] is 01. The default [datafile] name is SYSNAM, and the default [datafile] extension is TXT. The [options] field indicates an initial command. The [options] accepted are "S" for START; "A" for ADD; "M" for MODIFY; and "F" for FIND.

The interpreter displays a sign on message:

```
DF11 INTERPRETER 2.n - ddmmmyy
```

The START and ADD commands place the data file in an "OPEN" mode. The data file must be placed in the "CLOSED" mode (e.g., by use of the END command), before another START or ADD command may be entered.

The interpreter will respond to the commands discussed below. A form number (in decimal) is optional in most of these commands; if it is omitted, the current form will be assumed. An error may occur if a form number is required and none is currently in use.

<u>Only the first letter of a command is required</u>; for example, "START 2" may also be entered as "S 2".

DATA ENTRY FLOW

DF11 <SYSNAM>[,datafile][;mode]

Enter
Operator
Commands

Operator
Input

Enter END
Operator Command

SYSNAM/TXT

5.1 The START Command

The:

```
START [nn]
```

command displays the following message:

```
START <filename/ext> ON DRn?
```

before any data is placed into the file, so that the operator can verify whether the filename, extension, and drive number are correct. If the name, extension, and drive are correct, a response of "Y" should be entered. The START command causes data to be placed at the beginning of the data file.

After a response of "Y" to the START message, if a form number was specified on the START command; or if a form is currently in memory; that form is "entered" - i.e., the form is displayed with the cursor at the first non-constant field.

After a response of "N" to the START message; or if no form number was specified on the START command and no form is currently in memory, the message:

```
SELECT DATA MODE
```

is displayed, and control is returned to the interpreter's monitor.

NOTE: The START command does not check for possibly valid data in the data file; care must be taken so that a possibly valid data file is not overwritten.

5.2 The ADD Command

If the data file already exists, the:

```
ADD [nn]
```

command positions to the end of any data already in the file. If a form is already loaded or a form number is specified in the command, the form will be entered after the data file is positioned. if there is no form in memory, control is returned to the Interpreter's monitor.

5.3 The CONTINUE Command

If the data file is open, and the interpreter is positioned in the midst of the file, the:

```
CONTINUE [nn]
```

command backspaces the data file one logical record,and reads forward until an end of file mark is found. Other action is identical to the ADD command.

5.4 The LOAD Command

The first form to be loaded may be specified along with the system name on the interpreter command line, as indicated above.

If no number is supplied at that time, form number one is assumed (SYSNAM01). If any other form is to be loaded (replacing any form currently in memory), the:

```
LOAD <nn>
```

command loads form named "SYSNAMnn/DFF" into memory. If a data file has been opened, the form is entered. If no data file has been opened, the message:

```
FILE CLOSED
```

is displayed and control is returned to the interpreter's monitor.

New forms may be loaded without disturbing the position of the data file. Each data record has associated with it the form number with which it was created so that subsequent modification can identify data generated by a particular form.

If the form is not on the diskette, the message "BAD FORM" will appear.

5.5 The DATA Command

The:

```
DATA
```

command places the data file in the data entry mode initially, or returns to the data entry mode from the interpreter's monitor. If no form is in memory or if the data file is not open, an error message is displayed and control returns to the interpreter's monitor. Data currently in memory will not be disturbed and will be displayed whenever the form is re-entered.

5.6 Revising an Existing Data File

5.6.1 The MODIFY Command

Any data record on a DF11 generated data file can be accessed for _review or correction_.The:

```
MOD [nn]
```

command 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 file. The file is searched for the first data record created by the current form. Once a record has been found, the data file is in an "open" mode and may be searched in a forward direction by pressing the read next record function key (9), or, from the monitor mode, by entering another MOD command. To access records already passed over, the rewind function key (7) rewinds the data file (as does the initial MOD command).

If the data file is in the ADD/START mode, the MOD command automatically writes an end of file mark on the data file.

During modification, a new form may be loaded (without disturbing the position of the data file) 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. Previously recorded data supercedes form constants, thus, the actual data from the file will be displayed, overlaying the form's constants (and changing its display, if different). However, the _form's constants will be set into the data record_ when the field is entered (as they are for new records).

Data in a field may be changed at this time by entering new data in the field. Pressing ENTER in the first column of a field leaves the data unchanged. The edit criteria and field programs associated with the fields are still in effect, and will be re-executed.

5.6.2 The FIND Command

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

```
FIND [nn]
```

command may be used. This command loads the specified form (if different from the current form) and displays the form so the operator may enter characters into any fields to use as a key in searching the file. All edit criteria are applied to fields (except field programs and required edit criteria) when setting up the match data.

When the data to be matched has been entered, the operator must remember to press the ENTER key after data has been entered in the last field of the search key before pressing the read record function key (9) to start the search. The interpreter will search the data file 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.

If a match is not found, the message:

```
END OF DATA
```

appears and control is returned to the interpreter's monitor.

The search may be terminated by pressing both the KEYBOARD and DISPLAY keys simultaneously. The operator may want to stop a search if, for example, the wrong system name was specified, the wrong form was specified, or the wrong match data was given for a FIND. Control will be returned to the interpreter's monitor.

5.6.3 Rewriting Existing Records

Data records are rewritten, in both FIND and MODIFY modes, by the use of the write record function key (.). If the record was fetched using the MOD command, the next data record will automatically be read and displayed. If the record was fetched by the FIND command, control is returned to the interpreter's monitor.

If no field needs to be changed, the next record can be fetched by pressing the read next record function key (9); note that any modifications made will be destroyed by the read function. The write record function key (.) must 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).

5.7 The BACKSPACE Command

In the ADD/START mode, the:

BACKSPACE

command backspaces the data file one logical record after writing an end of file mark on the data file and placing the data file in the MODIFY mode.

In the MODIFY mode, the BACKSPACE command backspaces twice and reads forward once under form number control; that is, if the record being read was not created by the current form, preceding records will be read until a form number match is found.

The backspace record function key (8) also executes a BACKSPACE command.

5.8 The REWIND Command

The:

REWIND

command rewinds the data file and positions to the first data record created by the form currently loaded. The data file may not be rewound if it is in the ADD/START mode.

The rewind data file function key (7) also executes the REWIND command.

5.9 The END Command

The:

END

command is used to write an end of file mark on the data file. Switching from ADD/START mode to MODIFY mode automatically writes an end of file mark on the data file. The END command is rejected in the MODIFY mode.

5.10 The OS Command

The:

```
OS
```

command is used to terminate exeution of DF11. No file mark is written on the data file. The DOS is reloaded.

5.11 The HELP Command

The:

```
HELP
```

command will display an explantion of interpreter commands and function keys. The data file must be closed before the HELP command is accepted. The HELP command overlays the current form in memory with a standard form whose name is DF11/OV1.

The DF11/OV1 form released with DF11 is a description of the DF11 command and function keys. It may be replaced by a form more closely related to the data entry system as seen by the operator.

Pressing the load next form function key (1) will load form one of the data entry system. Pressing the monitor function key (4) will return to the interpreter's monitor.

5.12 Data Entry Action

In the data entry mode, 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. Data set by the SEMI-CONSTANT command at form generation time is displayed and the cursor is 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 entered 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 TYPE edit criteria for the field (alpha, numeric or mixed) a BEEP is sounded and the cursor does not advance.

A BEEP is sounded if a key is pressed and the Interpreter is not ready to accept a character because of disk activity.

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

When the interpreter detects an error in a field, it places the cursor at the beginning of the field just entered and causes the processor to BEEP. The cursor does not advance to the next field. The unacceptable data is not set in the data area in memory, but still appears on the screen. If the operator decides to tab past the field, the last accepted data (blank if none has been entered) is displayed.

5.13 Interpreter Function Keys

The ENTER key is used as a forward tab key and the backspace field function key (3) is used as a backward tab key. Forward tabbing past required fields is not permitted. Note that alpha/numeric editing occurs as data is being entered into the field. 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). Field programs are not executed until all other editing has been performed successfully.

5.13.1 The Form Data Duplicate Function Key (0)

Once a form has been completed, the data is transferred to the OUTPUT buffer from which it is written to the data file. The OUTPUT buffer is available to the operator for form data duplication by means of the form data duplicate function key (0). If no previous record has been written, or if the preceding record was created by a different form, the results of pressing the form data duplicate function key (0) are undefined.

5.13.2 The Load Next Form Function Key (1)

The next form (specified by the linkage information in the current form) will be brought into memory when the load next form function key (1) is pressed. The current data record must be recorded, either under program control, or by use of the write record function key (.), prior to loading the next form, since pressing the load next form function key (1) does not write the data record, but instead clears any data in memory.

5.13.3 The Backspace Field Function Key (3)

The backspace field function key (3) is used to retreat from a field to the previous field. No indication is given to field programs that the backspace field function has been executed.

5.13.4 The Return to Monitor Function Key (4)

Whenever it becomes necessary to execute one of the interpreter "commands" while entering data into a form, the operator must press the return to monitor function key (4) to return control to the interpreter's monitor. Only then may the command be executed.

5.13.5 The Form Data Erase Function Key (6)

The form data erase function key (6) clears the entire data area (without writing it to the file) and redisplays the cleared form. No indication is given to field programs that the Data Erase function has been executed.

5.13.6 The Rewind Data File Function Key (7)

The rewind data file function key (7) is acceptable only in MODIFY mode. The rewind data file function key (7) rewinds the data file and positions to the first data record created by the currently loaded form.

5.13.7 The Backspace Record Function Key (8)

If the data file is in ADD/START mode, the backspace record function key (8) causes the interpreter to write an end of file mark on the data file, place the data file in MODIFY mode, and display the next preceding data record written using the current form.

If the data file is in the MODIFY mode, the backspace record function key (8) causes the interpreter to display the next preceding data record written using the current form.

5.13.8 The Read Record Function Key (9)

The read record function key (9) is acceptable only in MODIFY mode. It causes the interpreter to search forward in the data file for the next record that was written by the current form.

5.13.9 The Write Record Function Key (.)

The write record function key (.) is used to write the current data record to the data file. If one or more required fields have not been completed when the write record function key (.) is pressed, the processor BEEPs and the cursor is placed at the first unfilled required field. No data is written to the file. If all required fields are completed, a data record will be written to the data file whenever the write record function key (.) is pressed. The data record is written even if only incomplete data has been entered. If an incomplete data record is written, it will contain ASCII 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 record has been written to the data file, the form will be displayed with all data fields cleared to null values (or to the form constants or semi-constants if any) ready for re-entry of data from the beginning. If, however, an auto-link is set when the write record function is executed, the data is written out and the linked form is automatically loaded and displayed.

5.14 Logical and Physical Data Records

The length of the data record generated during data entry is determined by the combined lengths of all data fields in the form (maximum 249 characters). The physical sector containing the data record also contains a form number (1 binary character biased by 4) and a pointer (1 binary character) which are transparent to all systems other than DF11 or programs especially designed to access DF11 data files. This means that editing or sorting or updating via DATABUS will remove the form numbers necessary to access these records under DF11.

The format of a logical 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 record terminator (015)

Logical records are packed into physical records. A logical record may span two physical records. The end of physical record character (003) is followed by pairs of binary numbers, representing the form number biased by 4 and starting disk buffer location of all logical records beginning in the physical record. Deleted records are entirely overwritten by the delete character (032), and their form numbers are set to zero. Incompletely filled data fields are filled with either spaces or zeros, depending on the field type edit criteria. Only 251 characters of the physical disk sector are used by DF11.

5.15 Data file and OVERFLOW

Approximately 840 sectors are available on a diskette which contain only an operating system. To insure proper termination in the event that a diskette becomes full of data, DF11 opens a file called OVERFLOW/SYS. Six sectors are allocated to this. When the last available sector on a diskette has been written to the data file, the following mesage is displayed:

```
DISKETTE FULL CONTINUE TO NEW DISKETTE (N,Y,
OR FORM NUMBER OF FINAL RECORD)?
```

If an "N" is entered, the last data record entered is written to the diskette, and an end of file mark is written on the data file. If a form number is entered, the last data record entered is written to the diskette, and that form is loaded and will be written to the data file, as a trailer record, before the end of file mark. If a "Y" is entered, the last data record entered and an end of file mark are written to the data file and the message:

```
REMOVE DISKETTE FROM DRIVE n; PLACE IN ENVELOPE;
ON WHICH DRIVE SHOULD THE FILE BE CONTINUED?
```

is displayed. When the drive number has been entered, a file whose name is the same as the original data file, but whose extension is TXn (where "n" is initially 1 and is incremented by 1 for every continuation) is created on that drive.

An attempt to continue a file whose extension is TX9 will cause the message:

```
FILENAME CANNOT BE CONTINUED
```

to be displayed.

CHAPTER 6. THE PRINT UTILITY

The print utility program (DF11PRT) may be used to print either DF11 data files or forms. The system name must be provided on the command line. If the system name includes a form number, that form will be printed. A parameter may also be placed on the command line following a semi-colon. A "D" parameter indicates the data file associated with the system should be listed; an "A" parameter causes all forms in the system to be printed.

6.1 Printing Disk Data Files

The data file may be printed either bv the "D" parameter or by answering "YES" to the question:

```
DO YOU WANT TO PRINT THE DATA FILE?
```

Each logical record of the file named SYSNAM/TXT is printed, 80 characters per line, on whichever printer (local or servo) is available. If a data record contains an embedded 015, it is interpreted as carriage return. If a data record contains an embedded 003, printing of the record will terminate prematurely.

6.2 Printing Forms

All forms associated with the system name will be printed if the "A" parameter is placed in command line. If the form number is specified along with the system name, that form will be printed. If no parameter or form number is supplied in the command line, the operator must answer "NO" to the message "DO YOU WANT TO PRINT THE DATA FILE". The message:

```
FORM NUMBER?
```

will be displayed. The desired form number should be entered. If "A" is entered, all forms in the system will be printed.

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, and the TYPE, JUSTIFY, REQUIRED, and PROGRAM edit criteria for each field.

CHAPTER 7. INFORMATION FOR THE PROGRAMMER

7.1 The Edit Table

7.1.1 Edit Table Format

For each field defined by a form, a six character set of edit criteria is generated. This entry describes the field in detail, as follows:

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 field is the number of characters the operator may enter -- from 1 to 80. This number is associated at execution time with the labels INPUT, OUTPUT and with field references in field programs.

The position in output 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's value is 0377.

The edit key is a combination of bits indicating the edit criteria set in the generator TYPE and REQUIRED passes. 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 character. The number of the last field in the screen image (the first is zero) is used to determine the length of the edit table. In addition, there is an 0377 stored after the last entry in the edit table.

7.1.2 Work Area

During data entry, the six character set of edit criteria 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 character set of edit criteria. The location "SAVFLD" contains the current field number.

7.1.3 Routines to Access the Edit Table

There are several subroutines available to access the set of edit criteria. "EDTPNT" is the most basic subroutine. This subroutine uses the value in the C-register to set the HL registers to the address of the corresponding set of edit criteria.

"MOVEDT" stores the field number at "SAVFLD", and moves the corresponding set of edit criteria to the work area and into the registers. It also positions the cursor to the field.

"NEXT" and "LAST" use the field number at "SAVFLD" to access the next or the preceding field. Both subroutines call "MOVEDT".

7.2 Structure of the Form in Memory

7.2.1 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, linked form number +2 if a link is set), and the variable "PAGE3" is the auto-link flag (0 or 0377).

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

The set of edit criteria is always referenced via the address pointer "SEDIT"; the requested field is always checked against the maximum field number, "EEDIT".

7.2.2 Data Buffers

The OUTPUT buffer is always in a fixed position "DATA" at the end of all form pointers. Its length is defined 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 form data duplication function key (0).

7.2.3 Form Image

The compressed form is stored beyond the two output buffers and it is referenced indirectly through the pointers starting at the label "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.

7.2.4 Edit Criteria Table

The edit criteria table is generated beyond the compressed form. The character immediately after the edit table terminator (0377) is available for field programs.

7.2.5 Field Programs

When programs are attached to the form, blocks starting at relocatable addresses are given absolute addresses based at the first available space after the form edit table (the program base address). Non-relocatable records from the field program (e.g. COMMON), are simply passed through to the form file.

7.3 Subroutines Available in the Interpreter

7.3.1 DOS Facilities Available

The DOS interrupt handler and disk I/O routines are available. INCHL, DECHL and BLKTFR are also present. See the DOS User's Guide for descriptions and locations of the various routines.

7.3.2 Keyboard Input Routine

The interpreter contains its own keyboard input routine which has two entry points. 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$", parameters are provided to permit keyin of twenty characters with no edit restrictions. _The input is always stored in TEMP_.

7.3.3 Display Routine

The display routine also has two entry points, "DSPLY$" and "DSPLY". If the display routine is entered at "DSPLY", the cursor position will be set to the bottom line of the screen and the screen will be rolled up after the message is displayed. The message must be terminated by an 015. If the display routine is entered at "DSPLY$", 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 appear only at the beginning of the message (causing the screen to rollup one line); and 011 followed by a count, which may appear anywhere in the message (indicating space compression). In addition, binary zeros are converted to underscores and spaces are _not_ displayed at all (i. e., the cursor is simply positioned to the right). The message being displayed is always expanded into TEMP.

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

7.3.4 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 character entry to a work area for easy referencing. It also saves the field number in the variable "SAVFLD".

7.3.5 String Arithmetic Package

The string arithmetic package used in DF11 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

The entry point for add is ADD$, for subtract is SUB$, for divide is DIV$, and for multiply is MUL$.

7.4 Assembly Language Interfacing and Overlays

7.4.1 Program Base Address

When the form generator outputs a form, it displays a message:

```
PROGRAM BASE ADDRESS mmmm
```

The value, mmmm, is the decimal starting address of the form's programs. This information is of particular interest if assembly language programs are to be included with the form. The technique for utilizing this information is:

1. Generate a form and record the program base address.
2. Generate and assemble the assembly language program set at the program base address.
3. Compute the length (in decimal) of the assembly language program.
4. Generate and compile the DF11 program with a labeled WORK statement the same size as the assembly language program. (It may be necessary to use two WORK statements since the maximum reservable amount is 245 characters.)
5. Rerun the form generator. Enter OLD to retrieve the form and OUT to write the form with the DF11 program attached.
6. Use the DOS "APP" command to attach the

assembly language program to the form:

```
APP <assembly>,<form>,<newform>
```

This form may now be used by the interpreter.

NOTE: The assembly program and DF11 form should always be appended in this order, since, during conversion to cassette systems, the extended interpreter and all subsequent code are replaced by the cassette extended interpreter.

7.4.2 External References

Facilities are provided in the DF11 language to reference points outside the program, locations which may be either in the interpreter itself or in a separately assembled assembly language program.

The EQU instruction assigns an address to a label which may then be referenced by any of the branching statements in DF11 (GOTO, CALL, etc.). If this facility is used, the assembler return instruction "RET" will return control to either the statement after a CALL or to the NEXT point in the interpreter.

7.4.3 Returning to the Interpreter

A table of interpreter entry points is provided so that these address may be accessed at the same point in future versions:

```
NEXT$     EQU    06400
AGAIN$    EQU    06403
STORE$    EQU    06406
END$      EQU    06411
WEOF$     EQU    06414
```

To return to a field program after being called, the assembly language should simply return, "RET". Otherwise, a jump to the appropriate exit routine will return control to the interpreter.

7.4.4 Interpreter Data Areas

Various interpreter data areas may be needed by the assembly language programs. The variable TEMP is the single item keyin buffer and it is this area which is accessed when "INPUT" is referenced in a field program. References to "INPUT" are compiled as an address of 06000 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: Referencing a field other than the current field does not change the number of the current field.

Several variables in the interpreter may be useful to the program. To access external data, i.e., data in the interpreter or created by an assembly language routine, first EQU a label, then REDEFINE the label, assigning it the proper length. For example, the current field number (in binary, starting at zero), is at location 06141. To test for field 5:

```
CURFLD EQU 6141
CURENT REDEFINE CURFLD,1,1
FLD5   WORK 04
TEST   IF CURENT EQUAL FLD5 THEN XXXX
```

When the operator presses the ENTER key in the first position of a field, the current data is at INPUT and is then passed to the field program. The variable at location 06140, SAVNUM, contains a flag which is 0 if no data was entered, and is non-0 if data was entered.

7.4.5 Loading the Assembly Language Program

Since the format of a form and that of assembly code is the same, an assembly language overlay may be loaded by assigning it a name of SYSNAMnn/DFF and then entering "LOAD nn".

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 the DOS.

APPENDIX A. SAMPLE PROGRAMS

```
               <_______   |______ |
SIZE           7          6       1
TYPE           N
JUSTIFY        R
REQUIRED                  P       P
PROGRAM        S
```

SAMPLE PROGRAM - MOVE SIGN FROM LEFT END TO RIGHT END

```
INSIGN     REDEFINE     INPUT,1,1
INREST     REDEFINE     INPUT,2,6
NXTFLD     FIELD        +1
SIGN       FIELD        +2
SPACE      WORK         " ";
MINUS      WORK         "-";
.
.
. INPUT TO KEYIN ONLY FIELD; MOVE
. SIGN AND STORE IN NEXT FIELD
.
.
S*         IF           NULL NE INPUT THEN MOVE1
           IF           NULL EQ NXTFLD THEN AGAIN
           NEXT
MOVE1      MOVE         INREST TO NXTFLD
           MOVE         INSIGN TO SIGN
           IF           MINUS EQ INSIGN THEN NEXT
           MOVE         SPACE TO SIGN
           NEXT
```

```
            |_____ |   |   |_____   |  =  |_____   |
SIZE        6      1   1   6        1     6        1
TYPE        D              D
JUSTIFY     R              R
REQUIRED                                  P        P
PROGRAM                             K
```

SAMPLE PROGRAM - ARITHMETIC OPERATIONS ON FIELDS WITH SIGN ON RIGHT

```
LFT        WORK         7;
LFTSIGN    REDEFINE     LFT,1,1
LFTVALU    REDEFINE     LFT,2,6
MID        WORK         7;
MIDSIGN    REDEFINE     MID,1,1
MIDVALU    REDEFINE     MID,2,6
SUM        WORK         7;
SUMSIGN    REDEFINE     SUM,1,1
SUMVALU    REDEFINE     SUM,2,6
ADD        WORK         "+";
SUB        WORK         "-";
MPY        WORK         "*";
DIV        WORK         "/";
SPACE      WORK         " ";
ZED        WORK         "0";
VALU1      FIELD        1
SIGN1      FIELD        2
OP         FIELD        3
VALU2      FIELD        4
VALU3      FIELD        6
SIGN3      FIELD        7
.
.
K*         MOVE         INPUT TO OUTPUT
           MOVE         VALU1 TO LFTVALU
           MOVE         SIGN1 TO LFTSIGN
           MOVE         VALU2 TO MIDVALU
           MOVE         INPUT TO MIDSIGN
           IF           OP EQ ADD THEN ADD1
           IF           OP EQ SUB THEN SUB1
           IF           OP EQ MPY THEN MPY1
           IF           OP EQ DIV THEN DIV1
           CHANGE       3
           AGAIN
ADD1       ADD          MID TO LFT GIVING SUM
```

```
          GOTO      DONE
SUB1      SUB       MID FROM LFT GIVING SUM
          GOTO      DONE
MPY1      MPY       MID BY LFT GIVING SUM
          GOTO      DONE
DIV1      DIV       MID INTO LFT GIVING SUM
DONE      MOVE      SUMVALU TO VALU3
          IF        SUMSIGN EQ ZED THEN BLANK
          MOVE      SUMSIGN TO SIGN3
          NEXT
BLANK     MOVE      SPACE TO SIGN3
          NEXT
```

```
                <______     |______
SIZE            6           7
TYPE            D
JUSTIFY         R
REQUIRED                    P
PROGRAM         C
```

SAMPLE PROGRAM TO COMPUTE CHECK DIGIT (MOD 10)

```
COMBO       WORK        "0000000";
CKWORK      REDEFINE    COMBO,1,6
CKDIG       REDEFINE    COMBO,7,1
NXTFLD      FIELD       +1
ONE         WORK        "1";
WEIGHT      WORK        "121212";
.
.
C*          MOVE        INPUT TO CKWORK
            SUB         CKDIG FROM CKDIG
C1          IF          COMBO CK10 WEIGHT THEN C2
            ADD         ONE TO CKDIG
            GOTO        C1
C2          MOVE        COMBO TO NXTFLD
            STORE
```

```
                MM   DD   YY          JULIAN
                |-   |_   |_          |-  |__
SIZE            2    2    2           2   3
TYPE            D    D    D
JUSTIFY         R    R    R
REQUIRED                              P   P
PROGRAM         G    H    J
```

SAMPLE PROGRAM TO CONVERT TO JULIAN DATE

```
ADDER          WORK             "000","031","059","090","120","151":
                .               "181","212","243","273","304","334"
LEAPYR         WORK             "76","80","84","88","92","96"
MONTH          WORK             "01","12"
DAY            WORK             "01","31"
HOLD           WORK             "000";
K1             WORK             "1";
K02            WORK             "02";
INDAY          FIELD            -1
INMO           FIELD            -2
JYR            FIELD            +1
JDAY           FIELD            +2
.
.
G*             IF               INPUT NIR MONTH THEN AGAIN
               STORE
.
.
H*             IF               INPUT NIR DAY THEN AGAIN
               STORE
.
.
J*             ALIGN            INMO TO HOLD
               LOOKUP           HOLD IN ADDER GIVING JDAY
               ADD              INDAY TO JDAY
               MOVE             INPUT TO JYR
               IF               INPUT NIT LEAPYR THEN STORE
               IF               INMO LE K02 THEN STORE
               ADD              K1 TO JDAY
               STORE
```

```
                |_____________________________
SIZE            30
TYPE            M
JUSTIFY
REQUIRED
PROGRAM         C
```

SAMPLE PROGRAM ILLUSTRATING CHARACTER CONVERSION

```
IN1         REDEFINE      INPUT,1,1
INMOVE      REDEFINE      INPUT,2,29
ALLOUT      DATA          1,30
WK1         REDEFINE      ALLOUT,30,1
WKMOVE      REDEFINE      ALLOUT,2,29
ASCII       WORK          "ABCDEFGHIJKL":
                          "MNOPQRSTUVWX":
                          "YZ0123456789"
EBCDIC      WORK          O301,302,303,304,305,306:
                          307,310,311,321,322,323:
                          324,325,326,327,330,331:
                          342,343,344,345,346,347:
                          350,351,360,361,362,363:
                          364,365,366,367,370,371
SINGLE      REDEFINE      EBCDIC,1,1
K29         WORK          "29";
COUNT       WORK          "00";
K00         WORK          "00";
K1          WORK          "1";
.
.
C*          MOVE          K29 TO COUNT
C1          CONVERT       IN1 BY ASCII AND SINGLE GIVING WK1
            MOVE          INMOVE TO INPUT
            MOVE          WKMOVE TO ALLOUT
            SUB           K1 FROM COUNT
            IF            K00 NE COUNT THEN C1
            NEXT
```

```
                 SHIPPED TO  |______________     SHIPPED FROM  |______________
SIZE                         15                                15
TYPE
JUSTIFY
REQUIRED
PROGRAM                      A                                 A

                    ADDRESS  |______________          ADDRESS  |______________
SIZE                         15                                15
TYPE
JUSTIFY
REQUIRED
PROGRAM                      A                                 B

                       DATE  |_  |_  |_                  DATE  |_  |_  |_
SIZE                         2   2   2                         2   2   2
TYPE                         D   D   D                         D   D   D
JUSTIFY                      R   R   R                         R   R   R
REQUIRED
PROGRAM                              C
```

SAMPLE PROGRAM ILLUSTRATING ENTERING FIELDS "OUT OF ORDER"

```
.
.
A*          CALL        SAVE
            CHANGE      +1
            NEXT
.
.
B*          CALL        SAVE
            CHANGE      7
            NEXT
.
.
C*          CALL        SAVE
            CHANGE      1
            NEXT
.
.
SAVE        MOVE        INPUT TO OUTPUT
            RETURN
```

```
                THIS IS THE FIELD
                |_______________
SIZE            15
TYPE
JUSTIFY         R
REQUIRED
PROGRAM         V
```

SAMPLE PROGRAM TO ILLUSTRATE MODIFY MODE VERFICATION

```
ONE         WORK          "1";
THREE       WORK          "3";
HOLD        WORK          "               ";
COUNT       WORK          "0";
.
.
V*          IF            RETRY EQ NULL THEN STORE
            IF            INPUT EQ OUTPUT THEN OK
            ADD           ONE TO COUNT
            IF            COUNT EQ THREE THEN TRY
            MOVE          INPUT TO HOLD
            AGAIN
TRY         IF            INPUT NE HOLD THEN ATTEMPT
            MOVE          INPUT TO OUTPUT
OK          SUB           COUNT FROM COUNT
            NEXT
ATTEMPT     SUB           COUNT FROM COUNT
            AGAIN
```

```
                  |______
SIZE              7
TYPE              N
JUSTIFY           R
REQUIRED
PROGRAM           A

                  |______
SIZE              7
TYPE              N
JUSTIFY           R
REQUIRED
PROGRAM           A

                  |_ ____
SIZE              7
TYPE              N
JUSTIFY           R
REQUIRED
PROGRAM           A

                  |______
SIZE              7
TYPE              N
JUSTIFY           R
REQUIRED
PROGRAM           A

                  |______
SIZE              7
TYPE              N
JUSTIFY           R
REQUIRED
PROGRAM           A

                  0000.00
SIZE              7
TYPE
JUSTIFY
REQUIRED          P
PROGRAM
```

SAMPLE PROGRAM - TOTAL ACCUMULATION (FIRST STYLE)

```
TOTAL       FIELD       6
.
.
A*          SUB         OUTPUT FROM TOTAL
            ADD         INPUT TO TOTAL
            STORE
```

```
                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          F

                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          F

                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          F

                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          F

                 0000.00
SIZE             7
TYPE
JUSTIFY
REQUIRED         P
PROGRAM
```

SAMPLE PROGRAM - TOTAL ACCUMULATION (SECOND STYLE)

```
BLANK       WORK         "            ";
SUM         FIELD        5
.
.
F*          SUB          OUTPUT FROM SUM
            ADD          INPUT TO SUM
            MOVE         INPUT TO OUTPUT
```

```
MESSAGE      BLANK
MESSAGE      SUM
NEXT
```

```
              |______
SIZE          7
TYPE          N
JUSTIFY       R
REQUIRED
PROGRAM       Z

              |______
SIZE          7
TYPE          N
JUSTIFY       R
REQUIRED
PROGRAM       Y

              |______
SIZE          7
TYPE          N
JUSTIFY       R
REQUIRED
PROGRAM       X

              |______
SIZE          7
TYPE          N
JUSTIFY       R
REQUIRED
PROGRAM       W

              0000.00
SIZE          7
TYPE
JUSTIFY
REQUIRED      P
PROGRAM
```

SAMPLE PROGRAM - TOTAL ACCUMULATION (THIRD STYLE)

```
ONE          FIELD          1
TWO          FIELD          2
THREE        FIELD          3
FOUR         FIELD          4
FIVE         FIELD          5
.
.
```

```
Z*          ADD           INPUT TO TWO GIVING FIVE
            GOTO          EXIT1
Y*          ADD           ONE TO INPUT GIVING FIVE
EXIT1       ADD           THREE TO FIVE
            GOTO          EXIT2
X*          ADD           ONE TO TWO GIVING FIVE
            ADD           INPUT TO FIVE
EXIT2       ADD           FOUR TO FIVE
            STORE
W*          ADD           ONE TO TWO GIVING FIVE
            ADD           THREE TO FIVE
            ADD           INPUT TO FIVE
            STORE
```

```
                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          P

                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          Q

                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          R

                 |______
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM          S

                 0000.00
SIZE             7
TYPE
JUSTIFY
REQUIRED         P
PROGRAM
```

```
SAMPLE PROGRAM  TOTAL ACCUMULATION (FOURTH STYLE)

ONE         FIELD        1
TWO         FIELD        2
THREE       FIELD        3
FOUR        FIELD        4
FIVE        FIELD        5
BLANK       WORK         "          ";
.
```

```
P*          ADD         INPUT TO TWO GIVING FIVE
            GOTO        EXIT1
Q*          ADD         ONE TO INPUT GIVING FIVE
EXIT1       ADD         THREE TO FIVE
            GOTO        EXIT2
R*          ADD         ONE TO TWO GIVING FIVE
            ADD         INPUT TO FIVE
EXIT2       ADD         FOUR TO FIVE
            GOTO        EXIT3
S*          ADD         ONE TO TWO GIVING FIVE
            ADD         THREE TO FIVE
            ADD         INPUT TO FIVE
EXIT3       MOVE        INPUT TO OUTPUT
            MESSAGE     BLANK
            MESSAGE     FIVE
            NEXT
```

```
                 0000.00
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM

                 0000.00
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM

                 0000.00
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM

                 0000.00
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED
PROGRAM

                 0000.00
SIZE             7
TYPE             N
JUSTIFY          R
REQUIRED         R
PROGRAM          T
```

SAMPLE PROGRAM - TOTAL ACCUMULATION, CHECKING AGAINST KEYED IN TOTAL

```
FIRST      FIELD        1
SECOND     FIELD        2
THIRD      FIELD        3
FOURTH     FIELD        4
TEMP       WORK         "0000.00";
SILVER     WORK         "CORRECT";
```

```
GOLD        WORK          "NOT CORRECT; 0000.00 IS CORRECT"
SHINE       REDEFINE      GOLD,14,7
.
.
T*          ADD           FIRST TO SECOND GIVING TEMP
            ADD           THIRD TO TEMP
            ADD           FOURTH TO TEMP
            IF            TEMP EQ INPUT THEN GREEN
            MOVE          TEMP TO SHINE
            MESSAGE       GOLD
            AGAIN
GREEN       MOVE          INPUT TO OUTPUT
            MESSAGE       SILVER
            NEXT
```

APPENDIX B. COMMANDS

GENERATOR:

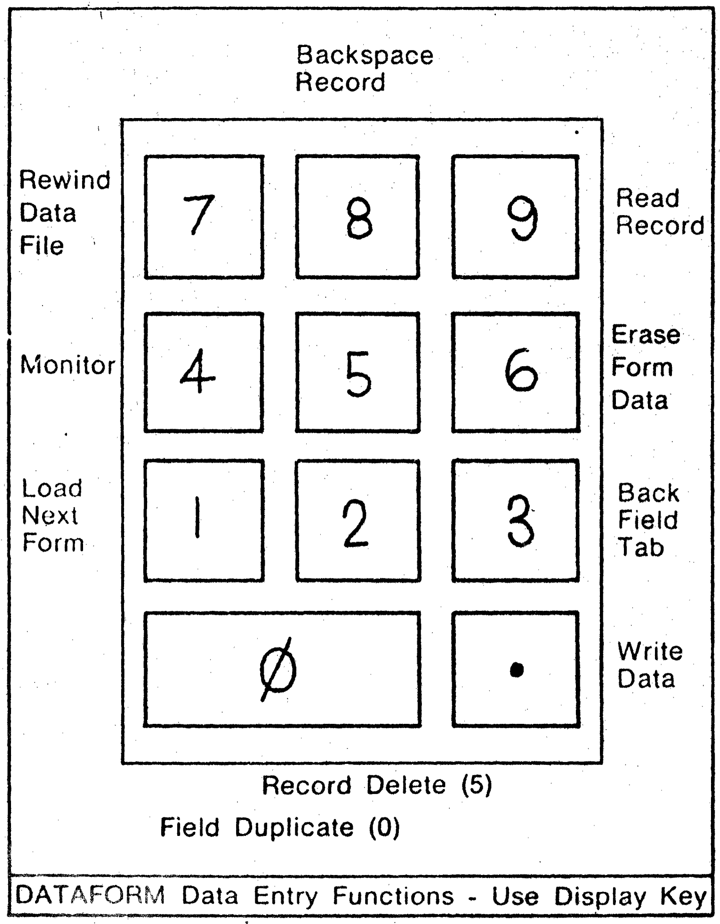
COMMAND	SECTION	MEANING
CONSTANT	3.7.3	set constants into the form
JUSTIFY		set filler and justification
LINK	3.7.5	define next form linkage
NEW	3.6	clear the work area for a form
OLD	3.10	load old form from front deck
OS		reload the DOS
OUT	3.8	write the current form to disk
PROGRAM	3.7.4	assign program letters to fields
REQUIRE	3.7.2	set required, fill controlled, or program reserved edit criteria
REVISE	3.9	revise the current form
SEMI-CONSTANT	3.7.3	set semi-constant data into the form
TYPE	3.7.1	set alphabetic or numeric edit criteria

INTERPRETER:

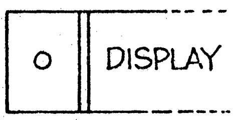
ADD	5.1.2	add to the end of a data file
BACKSPACE	5.5.1	backspace a record on data file
CONTINUE	5.1.3	add to the end of a data file if the file is already open
DATA	5.3	switch to data entry mode
END	5.6	write an end of file on the data file
FIND	5.4.2	search for matching data record
HELP		display instructions
LOAD	5.2	load the specified form
MODIFY	5.4.1	modify data records
OS	5.7	reload the DOS
REWIND	5.5.2	rewind data file
START	5.1.1	initialize a data tape

APPENDIX C. INTERPRETER FUNCTION KEYS

Mode	Key	Function
All Data Entry	DISPLAY/4	return to monitor
	DISPLAY/.	write data record or rewrite it
	DISPLAY/3	backward tab
	DISPLAY/5	delete record
	DISPLAY/6	erase form data
	DISPLAY/1	load next form
	DISPLAY/0	duplicate form data
Modify and Find Only	DISPLAY/7	rewind data file
	DISPLAY/8	backspace record
	DISPLAY/9	read record



This overlay is actual size. You can copy the page and cut out the template.



DATA ENTRY COMMANDS VIA NUMERIC KEYBOARD

APPENDIX D. FORM GENERATOR FUNCTION KEYS

The form generator has a set of special functions available in the image generation mode only. When the DISPLAY key is pressed, the number pad characters become function keys. The following functions are available:

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

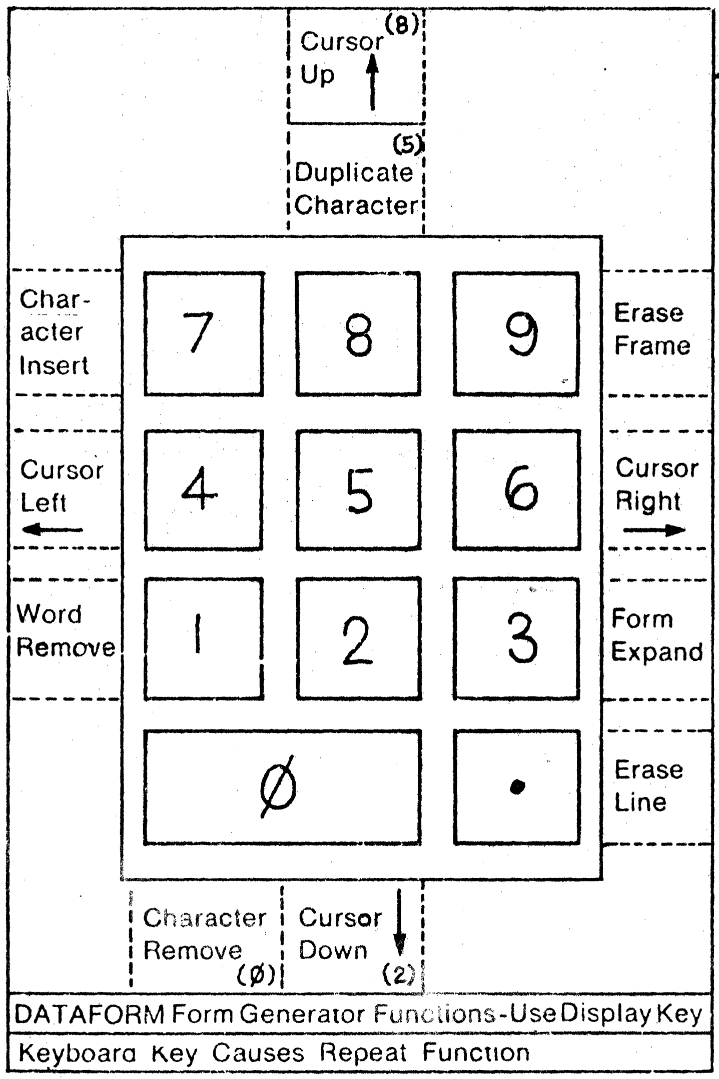
The BACKSPACE key and the cursor left function key have the same function. Backspacing from column 1 back to column 80 is permitted. All cursor movement with the special function keys is non-destructive.

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

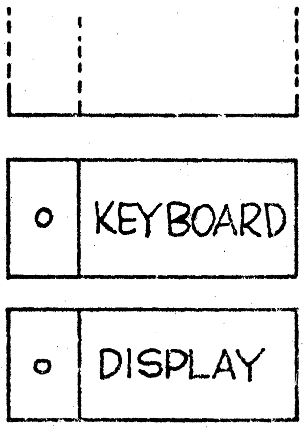
The KEYBOARD key acts as a repeat key for all characters and for most function keys.

The CANCEL function key returns to the form generator's monitor. The ENTER key places the cursor at the beginning of the next lower line.

NUMBER PAD OVERLAY



this overlay is actual size. You can copy the page and cut out the template.



FORM GENERATOR COMMANDS VIA NUMERIC KEYBOARD

APPENDIX E. FORM GENERATOR TYPE, JUSTIFY AND REQUIRE EDIT CRITERIA

TYPE	MEANING
A	Alpha (A - Z and space)
D	Digit (0 - 9)
M	Mixed alpha and numeric
N	Numeric (0 - 9, decimal point, and leading minus)
O	Numeric, minus overpunch
S	Shift key inversion
CANCEL	Clears edit criteria

Numeric fields are limited to 16 places of significance to the left and 8 places to the right of the decimal point.

JUSTIFY	MEANING
J	Right justify
Z	Zero fill
R	Zero fill Right justify

REQUIRE	MEANING
R	Required (1 character necessary)
F	Fill controlled (all characters necessary) (ENTER key allowed only to bypass field)
B	Both fill controlled and required
P	Program reserved (no keyin)
S	Required and program reserved (field is checked prior to write)
K	Keyin continuous
X	Keyin continuous and required

APPENDIX F. ALPHABETICAL LISTING OF STATEMENT TYPES

NAME	SECTION
ADD	4.6.2
AGAIN	4.7.1
ALIGN	4.6.1.1
BACKSPACE	4.6.4.9
BEEP	4.6.4.1
CALL	4.6.5.2
CHAIN	4.6.4.2
CHANGE	4.6.6
CLOSE	4.7.2
COMMON	4.5.3
CONVERT	4.6.1.2
DATA	4.5.1
DELETE	4.6.4.10
DIVIDE	4.6.2
END	4.7.3
ENTRYMODE	4.6.7
EQU	4.5.5
FORMSHOW	4.6.4.3
FIELD	4.5.7
FIELDNO	4.6.1.3
GOTO	4.6.5.1
IF CK10	4.6.3
IF CK11	4.6.3
IF INT	4.6.3
IF NIT	4.6.3
IF INR	4.6.3
IF NIR	4.6.3
IF EQ	4.6.3
IF NE	4.6.3
IF GE	4.6.3
IF LE	4.6.3
IF GREATER	4.6.3
IF LESS	4.6.3
INPUT	4.7.4
LOOKUP	4.6.1.4
MESSAGE	4.6.4.4
MODIFYMODE	4.6.7
MOVE	4.6.1.5
MULTIPLY	4.6.2
NEXT	4.7.5

APPENDIX G. INTERPRETER FLAG ADDRESSES

LABEL	LOCATION	DESCRIPTION
TEMP	06000	'INPUT' buffer
COLUMN	06130	edit entry - horizontal position
LINE	06131	vertical position
LENGTH	06132	field length
PSN	06133	position in OUTPUT
EDTKEY	06134	edit criteria
USER	06135	program letter
SAVNUM	06140	number chars entered (0 is ENTER pressed)
NEWOLD	06143	I/O mode/status word
FORMNO	06146	current form number +4 (in binary)
CURT	06156	address of next Dataform instruction
ADFLAG	06306	=0 if START, #0 if ADD
NEXTF	06161	Link form number +4 (in binary)
PAGE3	06162	Auto link flag
BASE	06163	Program base address
NEXT$	06400	Transfer to NEXT
AGAIN$	06403	Transfer to AGAIN
STORE$	06406	Transfer to STORE
END$	06411	Transfer to END
WEOF$	06414	Transfer to CLOSE
ERASE$	06417	Erase function key
DELET$	06422	Delete function key
ENTER$	06425	Re-enter form

APPENDIX H. COMPILER ERROR MESSAGES

NAME REQUIRED

The name of the program source file must be typed in the intial command line.

BAD LABEL INITIATOR

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

INVALID OCTAL

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

ILLEGAL OPERATOR

Something other than the accepted statement types was the first nonblank symbol after column 1 (or after the label, if one exists).

NUMBER FROM 1-249 EXPECTED

The indicated symbol is non-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 acceptable.

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 246, excluding the pre-defined labels. All labels after this maximum is reached are ignored.

COMMON LIMIT EXCEEDED

The COMMON block may not exceed 100 characters or the RESERVE amount. Anything defined as COMMON after this length will not be accepted.

COMMON PRECEEDS RESERVE

A RESERVE statement was encountered after a COMMON statement. Since RESERVE changes the starting address of COMMON, the RESERVE statement must come before all COMMON statements.

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.

COMMON SYSTEM ERRORS

FILE MISSING or FORM MISSING

The form number specified is not present as SYSNAMnn/DFF.

In the interpreter, this message may mean that the next form specified (in the current form's link) is not present, or that a command assumes that there is a form in memory (e.g. DATA) and none is loaded.

NAME REQUIRED

The initial command line did not include the system name or form name required by the program.

ILLEGAL DEVICE SPECIFICATION

The initial command line included a disk drive specification which was improperly formatted.

BAD NUMBER

The form number may have been omitted, out of range (1-99), non-numeric, or, the form specified is not in the disk directory as SYSNAMnn/DFF. 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.

PRINT UTILITY

BAD SYSTEM NAME

The name appearing on the command line was greater than 6 characters in length.

NO PRINTER

No printer is connected or turned on.

GENERATOR ERRORS

BAD FORM NAME
The form name specified in the command line did not end with a two digit number.

BAD FORM
The form in memory cannot be written out, or have any pass except REVISE executed, because of some error condition.

NO FIELDS DEFINED
Every form must contain at least one field (this field may be a keyin only field).

NO ROOM FOR CONSTANTS
Constants and semiconstants can only be assigned to fields of a form which were initially defined using the underscore (as opposed to the caret). This message is displayed if no constants can be assigned.

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 249 DATA
During image generation more than 249 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 are 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.

PROGRAM BASE ADDRESS XXXXX
This is the decimal Address of the first location in user space available for program code.

PROGRAM x MISSING

A program specified in the program pass is not contained in the program file (or there is no program file at all).

INTERPRETER ERRORS

Continuous Beeping during data entry
: An unacceptable 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.

SELECT DATA MODE
: No START, ADD, MOD or FIND command has been executed.

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

DATA FILE OPEN
: An open type operation was attempted before ending the current data file.

DATA FILE CLOSED
: A close type operation was attempted before opening the current data file.

NO FIELDS
: A form with no fields has been loaded.

NO LINK SET
: The operator attempted to load the linked form and no link was set.

ILLEGAL OP CODE
: An unacceptable DF11 op code was encountered during the execution of a field program.

APPENDIX I. USER SPACE REDUCTION TECHNIQUES

1. Use carets (^) in field definitions (remember they are compressed in the form image (not the data record) while underscores (_) are not).

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

3. 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.

4. Use subroutines to perform repeated operations.

5. 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).

6. Use INPUT, OUTPUT and RESET to generalize programs and thus avoid duplication of code.

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

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

9. Use LOOKUP instead of CONVERT to save one of the tables.

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

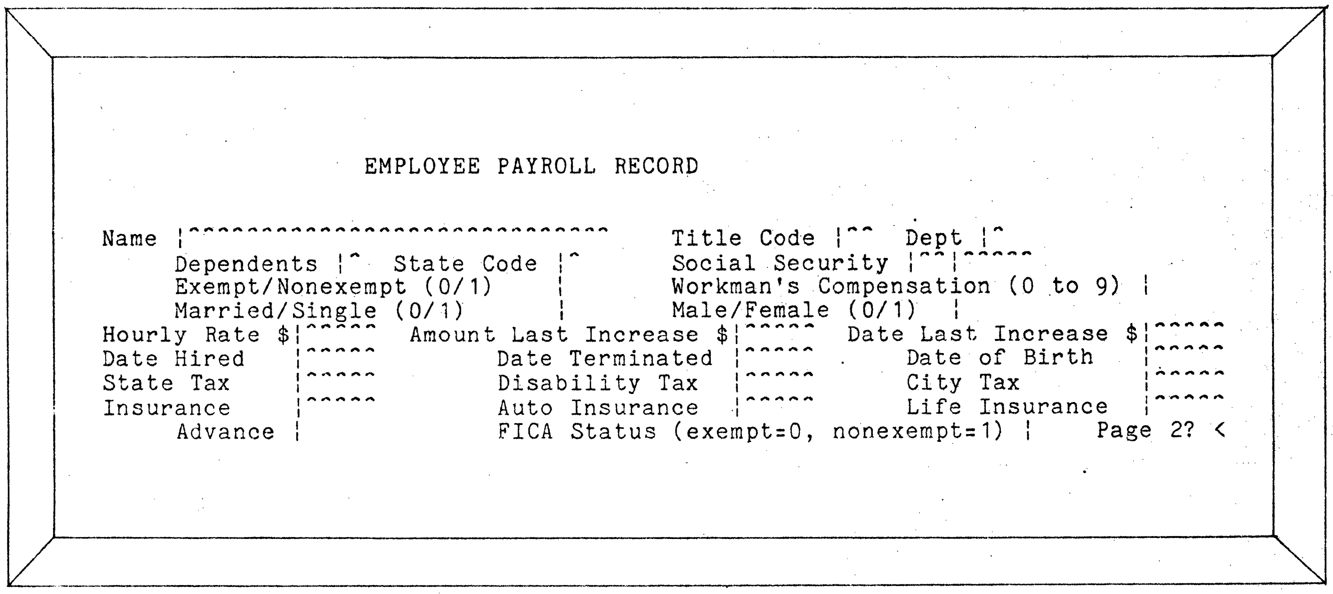
11. Execute all programs on last field if possible to save NEXT and STORE instructions.

12. Avoid CHANGE/SHOW/CHANGE as a series of instructions. Keep in mind that fields declared "program reserved" will show up on the screen in their sequence although the operator cannot keyin to them.

APPENDIX J. SAMPLE FORM GENERATION

Sample Form -- During NEW or REVISE Pass

Form text, data, and keyin only field definitions are set in either the NEW or REVISE pass. 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.

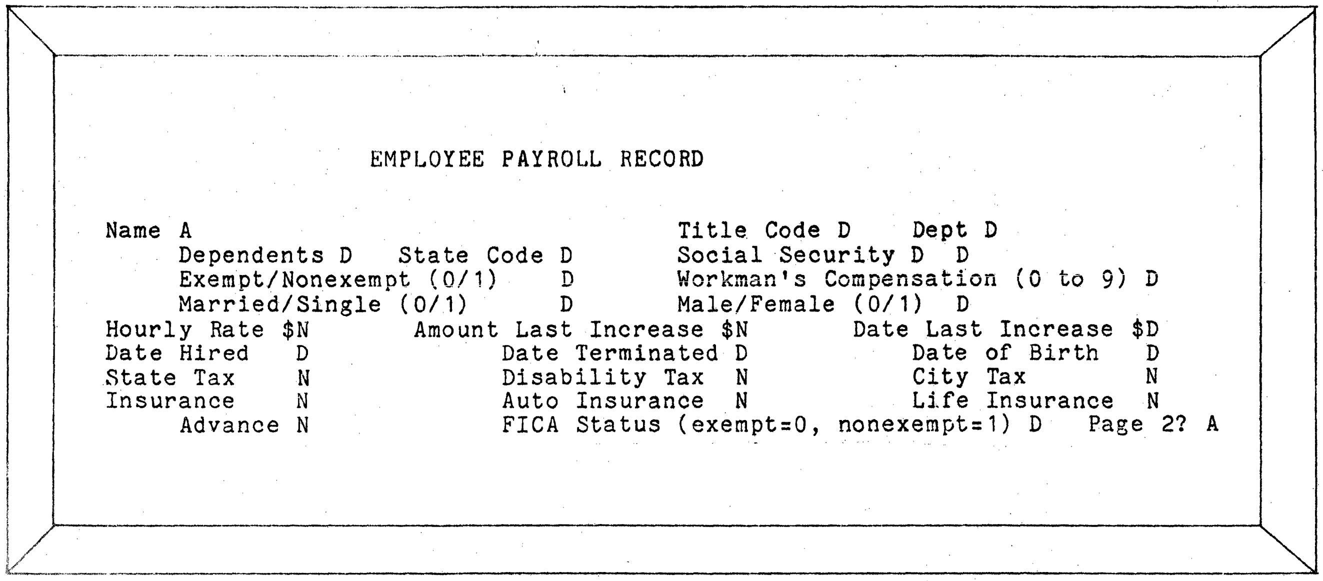


```
                  EMPLOYEE PAYROLL RECORD

Name ¦^^^^^^^^^^^^^^^^^^^^^^^^^^^^^    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 TYPE Pass

The field type edit criteria are set in the TYPE pass. Edit criteria will not be displayed during data entry.

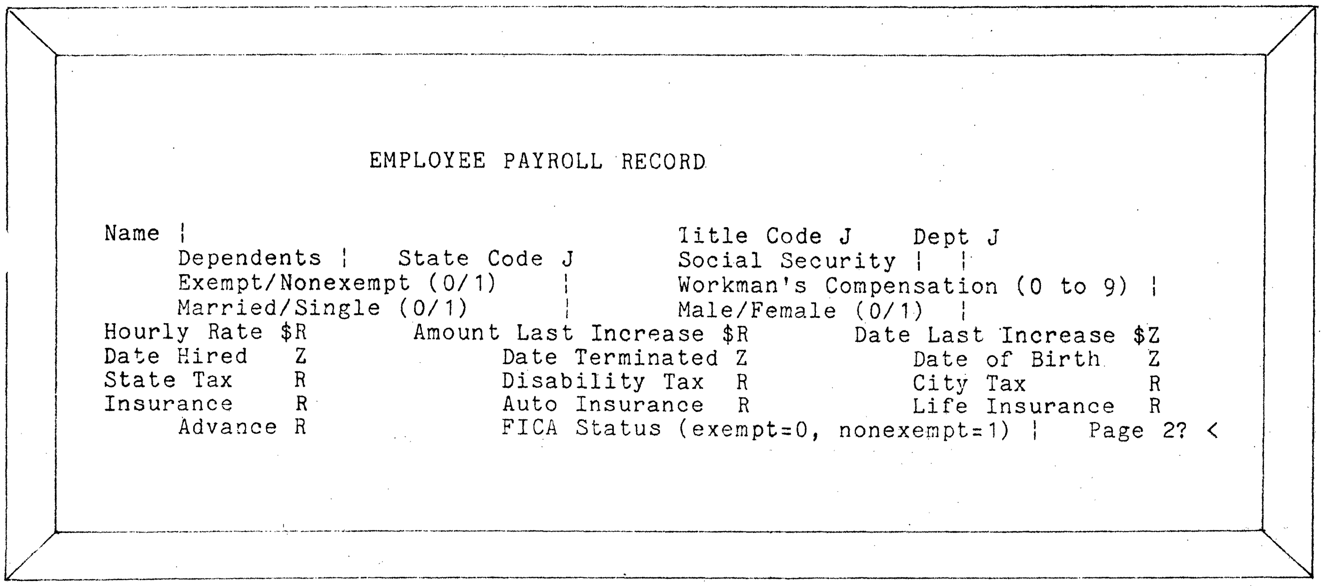


```
                    EMPLOYEE PAYROLL RECORD

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

Sample Form -- During JUSTIFY Pass

Right justification and field fill character are set in the Justify pass. Edit criteria will not be displayed during data entry.

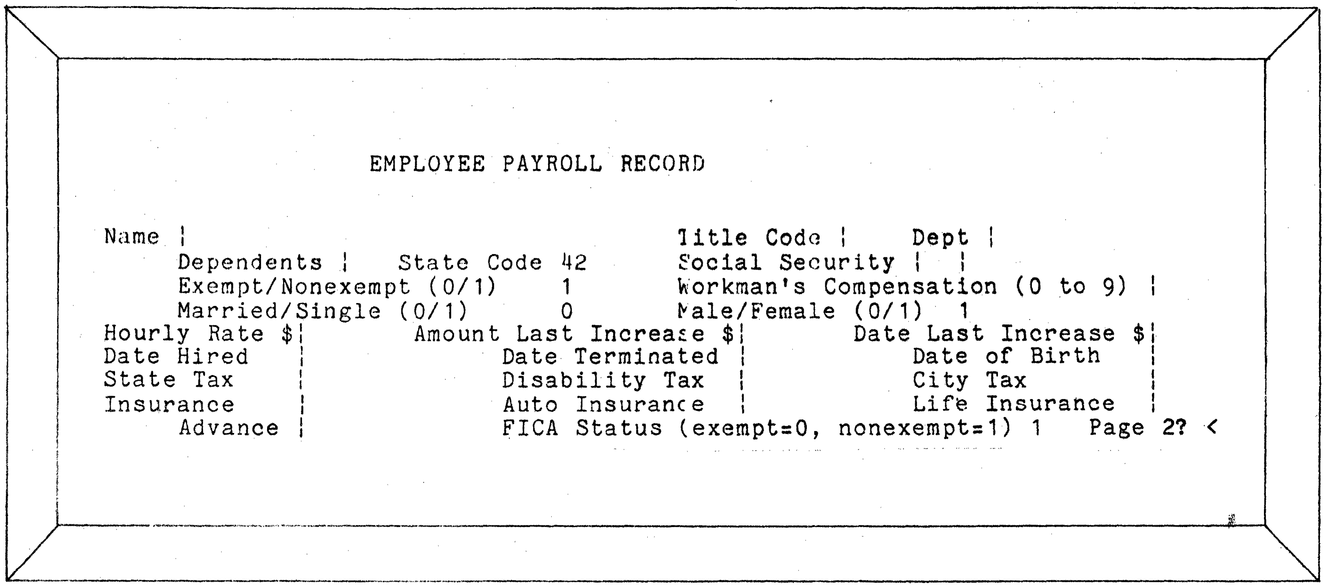


```
                 EMPLOYEE PAYROLL RECORD

Name ¦                                    Title Code J    Dept J
    Dependents ¦   State Code J           Social Security ¦  ¦
    Exempt/Nonexempt (0/1)    ¦           Workman's Compensation (0 to 9) ¦
    Married/Single (0/1)      ¦           Male/Female (0/1)  ¦
Hourly Rate $R        Amount Last Increase $R         Date Last Increase $Z
Date Hired   Z             Date Terminated Z              Date of Birth   Z
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) ¦    Page 2? <
```

Sample Form -- During SEMI-CONSTANT Pass

Several fields are preset to commonly entered values in the SEMI-CONSTANT pass. These may be accepted or rejected by the operator during data entry. The CONSTANT pass looks the same; however, constants may not be rejected during data entry.

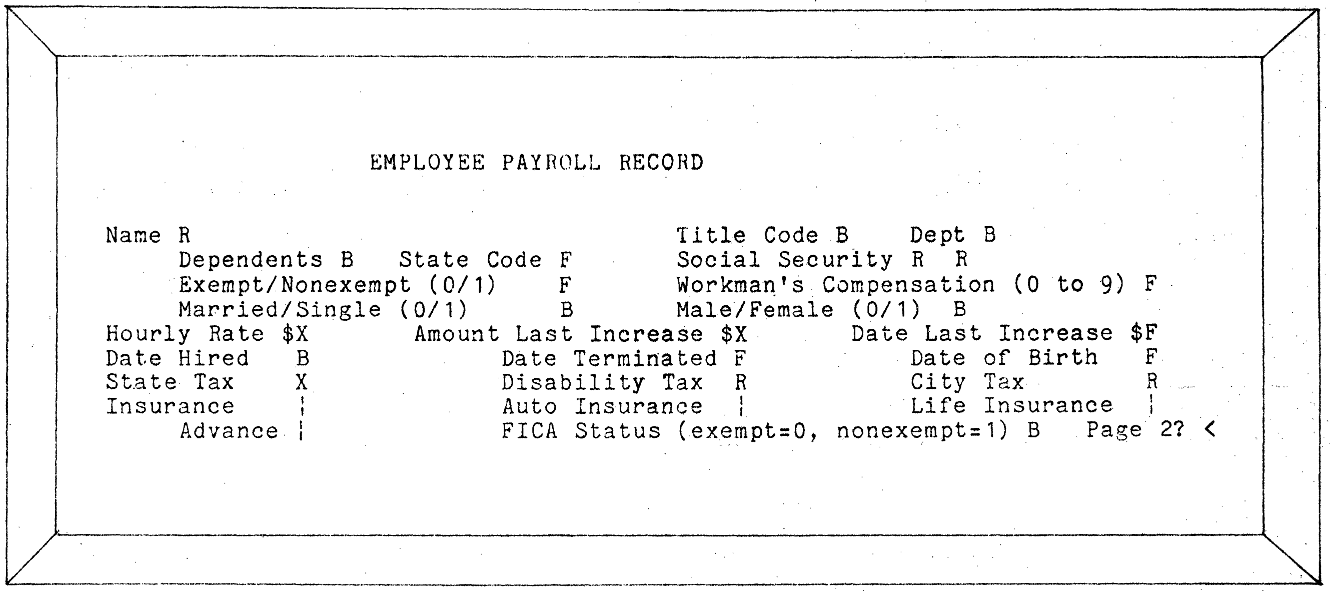


```
                     EMPLOYEE PAYROLL RECORD

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   Page 2? <
```

Sample Form -- During REQUIRED Pass

Required, fill controlled, and program reserved edit criteria are set in the REQUIRED pass. Edit criteria will not be displayed during data entry.

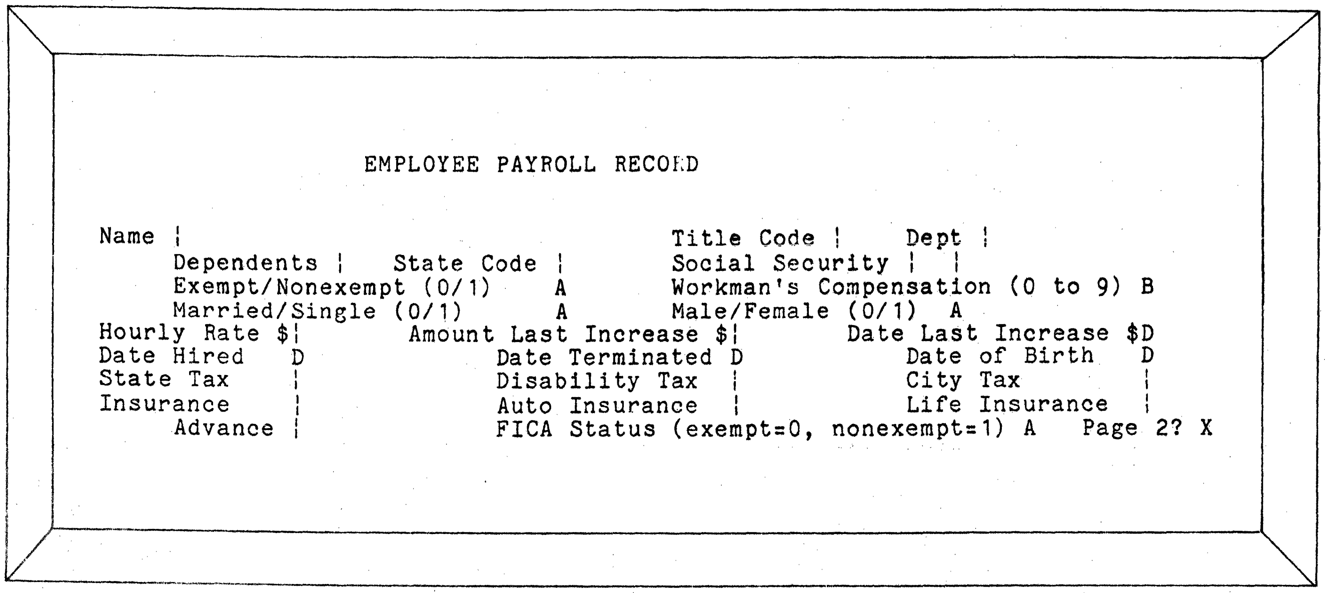


```
                  EMPLOYEE PAYROLL RECORD

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

Sample Form -- During PROGRAM Pass

Field program names are set in the PROGRAM pass. Program "A" checks range 0-1; "B" checks range 0-9; "D" checks for valid dates; and "X" checks for a "Y" or "N" to determine if another form should be loaded. Program names will not be displayed during data entry.



```
                  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
```