CTOS DATABUS User's Guide

August 1973

Model Code No. 50003

DATAPOINT CORPORATION

The Leader in
Dispersed Data Processing

COPYRIGHT© 1975 BY DATAPOINT CORPORATION. PRINTED IN U.S.A.

CTOS DATABUS

USER'S GUIDE

AUGUST 1973

TABLE OF CONTENTS

APPENDIX

INTRODUCTION

DATABUS, the Datapoint Business Lanugage, is a family of high-level programming languages designed especially for the Datapoint 2200 and its peripherals.

Unlike conventional small computers, which are built and shipped with little knowledge aforehand of what data processing devices will be attached to them, each Datapoint 2200 computer leaves the factory with at least video display, keyboard, dual cassette tape drives and a variable quantity of solid-state memory. This concept allowed the Datapoint systems programmers to construct a high-level language that could take full advantage of the built-in peripherals that are part of every Datapoint 2200.

The language is especially useful in commercial environments where jobs must be written quickly. The programmer may select the DATABUS language which contains features he will need to best accomplish the task. DATABUS 2, for instance, would be considered the best version to do extensive character manipulation, while DATABUS 3 contains features for data communications. For applications with limited memory size, DATABUS 4 will fit into only 4K of memory. In any case, data tapes generated are compatible between DATABUS versions.

This means that a DATABUS 2 program may generate a cassette data tape and a DATABUS 3 program can transmit that data to another 2200 in a location. All communications operations are handled in the language with even error checking handled automatically.

DATABUS will prove a useful and easily learned language for systems programming Datapoint users who are beginning implementation.

HOW TO USE THIS MANUAL

This manual is a complete reference to the DATABUS programming languages, 1 through 6. It is not, by any means, a textbook to learning the DATABUS language. A programmer who has had substantial background in COBOL, RPG, and other business-oriented language will soon feel familiar with DATABUS simply by reading the instruction set and referring to the examples at the rear of the manual.

Programmers who are relative newcomers to this type of language would do well to read Introduction to DATABUS, a comprehensive self-tutoring guide beginning with the fundamentals of system programming. A copy may be obtained through any sales office or by writing the home office.

CURRENT RELEASES OF DATABUS

The most recent releases and updates of the DATABUS program family are listed in the Appendix of this manual. Each printing of this manual reflects the releases in use. If in doubt, contact your local sales office or the Documentation Department in the San Antonio home office.

OTHER RELATED SOFTWARE

DATABUS versions 1-6 run under the Cassette Tape Operating System (CTOS) and separate documentation is available for this Operating System.

Additionally, the source code for a Databus program is generated on a cassette by use of the General Purpose Editor program (GEDIT). GEDIT is also documented in a separate manual.

Both CTOS and GEDIT documentation may be obtained through the local or home office. The current releases of these programs are also listed in the Appendix.

CTOS DATABUS

1.0 STATEMENTS

There are three basic types of statements in CTOS DATABUS: comment, data definition, and program execution. Comment lines begin with a period and may occur anywhere in the program. Comments are most useful in explaining program logic and subroutine function and parameterization to enable someone reading through the program to understand it more easily. Data definition statements must occur before any program execution statements and are used for setting up all the variables in the program. All data definition statements must have unique labels. Program execution statements must appear after any data definition statements and may or may not have labels. The labels on program execution statements may be the same as labels on the data definition statements. Program execution always begins with the first executable statement. The following are examples of CTOS DATABUS statements.

```
NAME    DIM 35
TITLE   INIT "TIME REPORT"
HOURS   FORM 5.2
TOTAL   FORM 10.2
RATE    FORM "2.50"
TAX     FORM "10.00"
.THIS IS A COMMENT
START   DISPLAY *H1,*V1,*EF,TITLE
        PREPARE 2
        KEYIN *H1,*V3,"NAME:",NAME
        KEYIN *H1,*V4,"HOURS:",HOURS
CALCR   MULT RATE BY HOURS
        ADD HOURS TO TOTAL
        SUB TAX FROM TOTAL
OUTPUT  PRINT "NAME:",NAME,*30,"RATE:",RATE;
        PRINT *40,"HOURS:",HOURS;
        PRINT *50,"TOTAL:",TOTAL
        WRITE 2,NAME,RATE,HOURS,TOTAL
        GOTO START
```

Labels for variables and executable statements may consist of any combination of up to six letters and numbers, but it must begin with a letter. The following are examples of valid symbols:

```
A
ABC
A1BC
B1234
ABCDEF
```

The following are examples of invalid symbols:

```
ABCDEFG   (too long)
HI,JK     (contains an invalid character)
3DAS      (begins with a number)
```

Statements other than comments consist of a label field, an operation field, and a comment field. The label field is considered empty if a space appears in the first column. The operation field denotes the operation to be performed on the following operands. In many operations, two operands may be connected either by an appropriate preposition (BY, TO, OF, FROM, or INTO) or a comma. One or more spaces should follow each element in a statement, except where a comma is used, in which case it must be the terminating character of the previous element and may be followed by any number (including zero) of spaces. The following are all examples of valid statements:

```
LABEL1  SUB  TWO FROM DIFF
LABEL2  SUB  TWO OF DIFF
LABEL3  SUB  TWO, DIFF            THIS IS A COMMENT
LABEL4  SUB  TWO,DIFF
```

Note that any prepositions may be used, even if it does not make sense in English. The following are examples of invalid statements:

```
LABEL1  SUB  TWO DIFF       (missing connective)
LABEL2  SUB  TWO ,DIFF      (space before comma)
```

Certain CTOS DATABUS statements allow a list of items to follow the operation field. In many cases, this list can be longer than a single line, in which case the line must be continued. This is accomplished by replacing the comma that would normally appear in the list with a colon and continuing the list on the following line. For example, the two statements:

```
PRINT A,B,C,D:
      E,F,G
PRINT A,B,C,D,E,F,G
```

will perform the same function. Note that the first entry of the continued line should not begin in the first column, the opcode field is the recommended place to begin the continued line.

2.0 DATA TYPES

There are three types of data used within the CTOS Databus language. They are numeric strings, character strings, and numeric indexes. The numeric variable arithmetic instructions are performed on numeric strings, the string instructions are performed on character strings, and the numeric index arithmetic instructions are performed on the numeric indexes. There are also instructions available to allow movement of numeric strings into character strings, character strings into numeric strings, numeric indexes into character strings, and character strings into numeric indexes. Each Databus version handles a subset of these three data types, and has a command set to handle the types of data it contains.

Numeric strings have the following memory format:

```
0200      1      2      .      3      0203
```

The leading character (0200) is used as an indicator that the string is numeric. The trailing character (0203) is used to indicate the location of the end of the string. Note that the format of a numeric string is set at definition time and does not change throughout the execution of the program. When a move into a number occurs from a string or differently formatted number, reformatting will occur to cause the information to assume the format of the destination number (decimal point position and the number of digits before and after the decimal point) with truncation occurring if necessary (rounding occurs if truncation is to the right of the decimal point). Character strings have the following memory format:

```
9      5      THE QUICK BROWN      0203
```

The first character is called the logical length and points to the last character currently being used in the string (K in the above example). The second character is called the formpointer and points to a character currently being used in the string (Q in the above example). The use of the logical and formpointer in character strings will be explained in more detail in the explanations of each character string handling instruction. Basically however, these pointers are the mechanism via which the programmer deals with individual characters within the string.

2.1 Variable Definition

Whenever a numeric or character string variable is used in a program, it must be "defined" at the beginning of the program using either the FORM, DIM, or INIT instructions. These instrucions reserve the memory space described above for the data variable whose name is given in the label field. Note that all variables must be defined before the first executable statement is given in the program and that once an executable statement is given no more variables may be defined. Numeric strings are created with the FORM instruction while character strings are created with the INIT or DIM instruction. The numeric indexes are set up for the user in the interpreters which handle them and do not need to be defined in the user's program.

2.2 Numeric String Variables

Numeric string variables are defined with the FORM instruction as shown in the following illustration:

```
EMRATE  FORM 4.2
XAMT    FORM " 382.4 "
```

In this example EMRATE has been defined as a string of decimal digits which can cover the range from 9999.99 to -999.99. The FORM instruction illustrated reserves space in memory for a number with four places to the left of a decimal point and two places to the right of a decimal point and initializes the value to zero. When the number is negative, one of the places to the left of the decimal point is used by the minus sign. XAMT, in the example, is defined with four places to the left of the decimal point and three to the right but with an initial decimal value of 382.400.

Care should always be taken when defining variables not to make them larger than will be needed for the values they will hold in the program. Making them larger than needed will set aside memory space that cannot otherwise be used and will reduce the overall space available to the program.

2.3 Character String Variables

Character strings are defined with either the dimension instruction, DIM, or the initialization instruction, INIT. The DIM reserves a memory space for the given number of characters, sets the length and formpointer to zero, and initializes all the characters to spaces. For example:

```
ANAME DIM 24
```

A character string can also be defined with some initial value by using the INIT instruction. For example:

```
TITLE INIT "PAYROLL PROGRAM"
```

initializes the string TITLE to the characters shown and gives it a logical length of 15. Note that in the case of strings, the actual amount of physical space reserved is three bytes greater than the number specified in the DIM or quoted in the INIT instruction (TITLE occupies 18 bytes in memory, 15 of which hold characters).

2.4 Numeric Indexes

To perform numeric operations in some of the interpreters, eight indexes have been set up. These indexes are referred to by instructions as I0 through I7. These do not need to be defined in the user's program and are initialized to zero at the beginning of every program. The indexes may be an integer value between 0 and 127 decimal.

Numeric indexes may only be used in the numeric index instructions and as indexes in LOAD, STORE, and BRANCH instructions. They may not be input and output between I/O devices.

3.0 INSTRUCTIONS

Every statement other than a comment must contain an instruction. There are nine classes of instructions to provide the basic types of operations the Datapoint 2200 must perform. They are:

DIRECTIVES - These instructions are basically instructions to the compiler. Directives define variables and establish their initial values and sizes. They may also establish the size of the user program, or tell the compiler to continue an instruction from one line to the next.

CONTROL - These instructions control the order in which a program is executed. They permit transfer of control from one part of the program to another depending on the results of other operations, stopping the program, or loading and running another program stored on the system tape.

STRING - These instructions perform the various string handling operations on character strings. The operations include string move, append, match, character match and move, and manipulation of the formpointer.

NUMERIC VARIABLE ARITHMETIC - These instructions perform the basic arithmetic operations on numeric variables, transfer of a value from one variable to another, and comparison of one variable to another.

NUMERIC INDEX ARITHMETIC - These instructions perform the basic arithmetic operations on string variables, comparison of indexes, and moving indexes to strings and back.

KEYBOARD, C.R.T., PRINTER INPUT/OUTPUT - These instructions perform the basic I/O functions to the mentioned devices.

CASSETTE TAPE INPUT/OUTPUT - These instructions perform the basic cassette tape handling functions for reading and writing tapes.

MAGNETIC TAPE INPUT/OUTPUT - These instructions perform the basic mag tape handling functions for reading and writing 7-Track and 9-Track magnetic tapes.

COMMUNICATIONS - These instructions provide the means to transmit and receive messages between Datapoint 2200's using 2200/Communication Adaptors.

Each Databus system contains a subset of these instructions to perform its functions. For example, Databus 1 contains the numeric variable arithmetic instructions but not the string variable instructions. Databus 2 contains both string and numeric variable arithmetic instructions but not the numeric index arithmetic or the communications. Databus 3 has the string, numeric index and communications facilities, but it does not have numeric variable arithmetic. Section 3, the Instruction Description Section, contains the entire CTOS Databus instruction set. See the individual Databus system sections for the instruction subset of the Databus system you are using.

The numbers in parentheses to the right of the instruction indicate which Databus version contains that instruction.

3.1 Directive Instructions

3.1.1 FORM (1)(2)

The FORM instruction defines the length and initial value of a numeric string variable. The FORM instruction must be used with a label which is used as the variable name throughout the program. The maximum length of a numeric string variable is 22 including the decimal point and minus sign.

Examples:

```
RATE    FORM "6.5"
AMT     FORM 6.2
ZERO    FORM 1
PI      FORM "3.14159"
```

If the numeric variable is defined with a quoted item, the same number of character positions are reserved in memory as are in the number between the quotation marks and the variable is initialized to the value given. In the above example RATE is dimensioned to a number with one place to the left and one place to the right of the decimal point, and initialized to a value of 6.5.

If the numeric variable is defined without quotes then the numbers that appear to the right and left of the decimal point specify how many positions to the right and left of the decimal point are reserved in memory. In the above example AMT reserves space in memory for a number with six places to the left of the decimal point and two places to the right of the decimal point and initializes the number to zero.

3.1.2 DIM (1)(2)(3)(4)(5)

DIM defines a character string variable, determines its physical length in memory, and initializes its logical length and formpointer to zero. The DIM instruction must be used with a label which is used as the variable name throughout the program. The maximum length of a character string variable is 127.

Examples:

```
REFLBL  DIM 60
XCODE   DIM 6
MAXLEN  DIM 127
```

3.1.3 INIT (1)(2)(3)(4)(5)

The INIT instruction is the same as the DIM instruction except that the initial value of the character string is established. This value may be initialized by either quoted strings or numerics for the old tape format interpreters. However in all the new tape format interpreters except DATABUS 3, only quoted strings are allowed to initialize strings. This is to insure that only legal printing characters will appear on the tape. The INIT instruction establishes physical and logical lengths that are equal, and initializes the formpointer to one.

Examples:

```
HDING  INIT "REORDER FORM"
DSFRM  INIT "NEXT ENTRY PLEASE",0101,10,015
```

The example HDING would be allowed in either old or new tape format interpreters. But the example DSFRM would not be allowed in all the new tape format interpreters since the numerics 0101, 10, and 015 are included.

3.1.4 Common Data Areas (1)(2)(3)(4)(5)

Since DATABUS has the provision to chain programs so that one program can cause another to be loaded and run, it is desirable to be able to carry common data variables from one program to the next. The procedure for doing this is as follows:

a. Identify those variables to be used in successive programs and in each program define them in exactly the same order and size and at beginning of each program. This is to cause each common variable to occupy the same locations in each program.

b. For the first program to use the variables, define them in the normal way.

c. For all succeeding programs place an asterisk in each FORM, DIM, or INIT statement as illustrated below to prevent those variables from being initialized when the program is loaded into memory.

Great care must be used when incorporating the feature into a program. An error in programming can produce strange results if a common variable is misaligned with respect to the variable in a previous program.

Example:

```
MIKE FORM *4.2
JOE  DIM  *20
BOB  INIT *"THIS STRING WON'T BE LOADED"
```

3.1.5 LENGTH (4)(5)

The LENGTH instruction defines the machine size to the compiler so that the user program may be tested for OVERFLOW in the machine size specified. A number must follow the LENGTH instruction. This number may be a 4, 6, 8, 12, or 16 corresponding to the memory size of the Datapoint 2200 being used. The LENGTH instruction must appear with the directives in a user program before the first executable instruction. This command is only available in Databus 4 and 5. If no length is specified in Databus 4, a 4K machine is assumed. In Databus 5, an 8K machine is assumed.

3.1.6 LINE CONTINUATION (2)(3)(4)(5)

The KEYIN, DISPLAY, PRINT, READ, WRITE, LOAD, STORE, and BRANCH instructions allow statements to be continued from one line to the next.

These instruction statements may be continued to the next line if a colon (:) is the terminating character of the instruction. The colon replaces the comma separating the last entry of the first line from the first entry on the second line. The first entry of the second line should begin in the instruction field. Examples of each are given in the instruction section.

3.2 CONTROL INSTRUCTIONS

3.2.1 GOTO (1)(2)(3)(4)(5)

The GOTO instruction transfers control to the program statement indicated by the label following the instruction:

```
GOTO CALC
```

causes control to be transferred to the instruction labeled CALC.

The GOTO instruction may be conditional, however, and the transfer of control occurs only if a specified condition is met. Seven possible conditions can be specified and are OVER, LESS, EQUAL, ZERO, EOS, TIME, and PARITY. The conditions result from previously executed instructions and reference should be made to the discussion on the various operations for their meaning (EQUAL and ZERO are two different names for the same flag).

In the example:

```
GOTO CALC IF OVER
```

control is transferred to the instruction labeled CALC if an overflow occurred with the last arithmetic operation, otherwise, the next instruction following the GOTO is executed.

The sense of the condition can be reversed as follows:

```
GOTO CALC IF NOT OVER
```

meaning control is transferred only if overflow did not occur.

3.2.2 CALL (1)(2)(3)(4)(5)

The CALL instruction is very similar to the GOTO instruction except that when a RETURN instruction is encountered after a transfer, control is restored to the next instruction following the CALL instruction. CALL instructions can be nested up to eight deep. That is, up to eight CALL instructions may be executed before a RETURN instruction is executed. Being able to call subroutines eliminates the need to repeat frequently used groups of statements, and may be made conditional as discussed in the GOTO instruction.

Examples:

```
CALL FORMAT
CALL XCOMP IF LESS
```

3.2.3 RETURN (1)(2)(3)(4)(5)

The RETURN instruction is used to transfer control to the location indicated by the top address on the subroutine call stack. This instruction has no operand field but may be conditional.

Examples:

```
RETURN
RETURN IF EQUAL
```

3.2.4 STOP (1)(2)(3)(4)(5)

The STOP instruction causes the program to terminate and return to the MASTER Program. If either tape deck is in write mode, that deck will automatically write an end-of-file mark before the program terminates.

The STOP instruction may be conditional as discussed under the GOTO instruction.

Examples:

```
STOP
STOP IF OVER
```

3.2.5 CHAIN (1)(2)(3)(4)(5)

The CHAIN instruction transfers control back to the operating system for the purpose of fetching and running another program on the operating system tape. There are two versions of the CHAIN command in CTOS Databus.

In Databus 1,2, and 3 the interpreter is cataloged on a CTOS tape. Therefore, chaining may be done by program name. The character string in the referenced variable is the name that appears in the CTOS catalog for the desired program. Any characters after the sixth will be ignored and blanks will be appended if less than six characters are in the variable. Note that the name used starts at the formpointer, so if in the following example NXTPGM's formpointer was 4, the CHAIN command would try to load the program named "ROL".

Example:

```
     NXTPGM INIT "PAYROL"
                  "
                  "
                 CHAIN NXTPGM
```

causes the program PAYROL to be loaded into memory and run.

In Databus 4 and 5, the interpreter is a LGO system, and the CTOS catalog is not on the system tape. Therefore, program chaining must be done by program file number. The character string in the referenced variable is the file number of the desired program. Only the first character of the string is used to determine a program number, and this number must be between 0 and 7 inclusive.

Example:

```
     NXTPGM INIT "3"
                  "
                  "
                 CHAIN NXTPGM
```

causes file 3 on the interpretive tape to be loaded into memory and run.

If the specified program is not on the interpretive system tape or if the program did not load successfully, the chain failure trap CFAIL will occur.

3.2.6 TRAP (1)(2)(3)(4)(5)

TRAP is a unique instruction because it does not take action at the time it is executed in the program but specifies that a transfer of control should occur later if a specified event occurs. For example:

```
     TRAP EMSG IF EOF2
```

specifies that control should be transferred to EMSG if an end-of-file mark is encountered on cassette deck two (front deck).

The transfer that occurs on all events except RING is like the GOTO instruction. On RING the transfer is like a CALL instruction, so that when a RETURN is executed after the transfer occurs, control is restored to the next instruction following the instruction executed when the ringing was detected.

The events that may be specified are:

EOF(n) - End-of-file mark on indicated device

EOT(n) - End-of-tape mark on indicated device

FORM(n) - Data of wrong type on indicated device - Old tape format

RFAIL(n) - Read failure on indicated device - New tape format

n = 1,2,3,4
1 = cassette deck 1
2 = cassette deck 2
3 = mag tape unit (adr = 264)
4 = mag tape unit (adr = 113)

CFAIL - Specified program not in catalog on chain instruction

RING - A ring detect for communications

On all events except RING, if the specified event occurs, but the trap is not set, the program will abort with the appropriate error message. In the case of RING, all ringing detected will be ignored if the trap is not set.

The ring trap is cleared after a transfer of control has been made.

3.2.7 TRAPCLR (3)

The TRAPCLR instruction clears the specified trap, so that a transfer of control will not occur should the specified event occur.

All events specified in the TRAP instruction discussion may be cleared by the TRAPCLR instruction. For example:

```
RNG TRAPCLR RING
```

specifies that if ringing is detected no transfer of control will occur.

3.2.8 BRANCH (1)(2)(3)(4)(5)

The BRANCH instruction transfers control to a statement specified by an index. In the Databus Interpreters which have numeric variables the index is a numeric variable. In the Databus Interpreters which do not have numeric variables, the numeric indexes, I0-I7, which have been set up in the Interpreters, may be used.

For example:

```
BRANCH N OF START,CALC,POINT
```

causes control to be transferred to the label in the label list pointed to by the numeric variable index N. (i.e. START if N=1, CALC if N=2, and POINT if N=3).

```
BRANCH I1 OF LIST,SUM,ENTER
```

causes control to be transferred to the label in the label list pointed to by the numeric index I1. (i.e. LIST if I1=1, SUM if I1=2, ENTER if I1=3). The index used may be any of the eight indexes I0 through I7.

If the index is negative, zero, or larger than the number of variables in the list, control continues with the following statement. Note that the numeric variable index is rounded to the nearest integer before it is used.

The BRANCH instruction statement may be continued to the next line if a colon (:) is the terminating character of the instruction. The colon replaces the comma separating the last entry of the first line from the first entry on the second line. The first entry of the second line should begin in the instruction field.

Example:

```
LABEL   BRANCH N OF LOOP,START,READ,WRITE:
        PRNT,END
        BRANCH I3 OF CONF,BUFF,DUMP,ROLL,NAME:
        SCAN,EXIT
```

3.2.9 ACALL (2)(5)

The Assembly Language Call Instruction allows the user to call assembly language subprograms to be executed outside of the interpreter. The assembly language programs should not overlay any of the interpreter or the Databus user area which calls it, unless the program reloads the interpreter or user program before returning, in which case the user program should be restarted.

Example:

```
ACALL 010000
```

calls a subprogram starting at location 10000 octal. The location to be called may be decimal or octal, but must be a number. The last statement in the subprogram executed should be a RET to return to the interpreter to resume execution of the Databus program. Only one entry in the stack must be preserved by the assembly subprogram, and this should be at the top of the stack upon return, i.e. no calls should be made within the subprogram without corresponding returns. If the stack is destroyed, however, the user may resume by jumping to the Databus Entry Point for the interpreter containing this instruction (03500 in Databus 2, 01400 in Databus 5).

There are two ways to load these subprograms into memory. One is to have all the subprograms on one or more LOAD & GO tapes and load them into memory before loading the LOAD & GO Databus Interpretive Tape.

The second method is to use the Databus CHAIN instruction. With this method, the first instruction of every program chained to must be a jump to the assembly subprogram entry point of the Interpreter (i.e., JMP 03500 in Databus 2, JMP 01400 in Databus 5). Jumping back to the Interpreter will cause execution of the next instruction after the chain. Using this method, all subprograms are cataloged on the Interpretive System tape and may be loaded in by the Databus user program.

ASSEMBLY PROGRAM FOR DATABUS CALL

```
        SET   016000
SUBR    JMP   03500 (01400)    RETURN TO DATABUS INTERPRETER

ENTRY   BEEP                   ASSEMBLY SUBPROGRAM
        HL    MESG
        LD    40
        LE    11
        CALL  DSP$
        RET

DSP$    EQU   016370 (05337)
MESG    DC    'ACALL TEST MESSAGE',0203


        END   SUBR
```

The above subprogram ENTRY would be called by ACALL 016003. The locations given are for Databus 2, those in parentheses are for Databus 5.

3.3 CHARACTER STRING HANDLING INSTRUCTIONS

Each string instruction, except LOAD and STORE, requires either one or two character string variable names following the instruction. (Note that the MOVE instruction is capable of moving strings to numbers, numbers to strings, numbers to numbers, strings to strings, indexes to strings, and strings to indexes. See sections 3.3.4, 3.4.5, and 3.5.4 for all descriptions of the MOVE instruction. In the following sections, the first variable will be referred to as the source string and the second variable will be referred to as the destination string.

3.3.1 CMATCH (2)(3)(4)(5)

CMATCH compares two characters, one taken from each of the source and destination operands. There are two versions of the CMATCH command in CTOS Databus.

In the Databus 2 and 3 CMATCH instruction, the characters to be compared may be from under the formpointer of a string variable, a quoted alphanumeric character, or a number. This number may be octal or decimal but it must have a value between 0 and 127 decimal.

An EOS condition occurs if the character is taken from a string which has a formpointer of zero, and no other conditions are set. Otherwise, the EQUAL and LESS conditions are set appropriately. The LESS condition is set if the second string character is less than the first string character.

Examples:

```
CMATCH XDATA TO YDATA
CMATCH Y,X
CMATCH "A",DOG
CMATCH DOG TO "B"
CMATCH CAT,0101
```

In the Databus 4 and 5 CMATCH, the first operand may be quoted alphanumeric or a numeric value less than 256. The second operand must be a string variable. The third operand must be a number. This number is used as the formpointer of the second string variable. The character under the formpointer of the second string is compared to the first operand character or value. If there is no third operand, the formpointer is assumed to be one.

The EOS condition is set if the destination string is null or if the formpointer specified is greater than the logical length of the string, and no other conditions are set. Otherwise, the EQUAL and LESS conditions are set appropriately.

Examples:

```
CMATCH "B" TO XDATA,3
CMATCH 0105,YDATA,15
CMATCH "C",STRING
```

3.3.2 CMOVE (2)(3)

CMOVE moves a character from the source operand to under the formpointer in the destination string. The character from the source operand may be a quoted alphanumeric, a number, or the character from under the formpointer of a string variable. If either operand has a formpointer of zero, an EOS condition and no transferral occurs.

Examples:

```
CMOVE XDATA,YDATA
CMOVE "A" TO CAT
CMOVE X,Y
CMOVE 0101 TO STRING
```

3.3.3 MATCH (2)(3)(4)

MATCH compares two character strings starting at the formpointer of each and stopping when the end of either string is reached. If either formpointer is zero, the MATCH operation will result in only clearing the LESS and EQUAL flags and setting the EOS flag. Otherwise, the "length" of each string is calculated to be LENGTH-FORMPOINTER+1 and the LESS flag is set if the destination string length is less than that of the source string. The two strings are then compared on a character-for-character basis for the number of characters equal to the lesser of the two lengths. If all the characters match, the EQUAL flag is set. If they do not match, the LESS flag's meaning is changed to indicate whether the numeric value of the destination character (in the character pair) is less than the numeric value of the source character (LESS flag set) or vice versa (LESS flag reset). Some examples and their results follow:

Source	Destination	Result
ABCDE	ABCD	EQUAL,LESS
ABC	Z	NOT EQUAL,NOT LESS
ZZZ	AAA	LESS,NOT EQUAL
ABC	ABC	EQUAL,NOT LESS
ABCD	ABCDE	EQUAL,NOT LESS

Examples:

```
MATCH A TO B
MATCH STR1,STR2
```

3.3.4 MOVE (2)(3)(5)

MOVE transfers the contents of the source string, starting from under the formpointer, into the destination string. Transfer into the destination string starts at the first physical character and when transfer is complete, the formpointer is set to one and the logical length points to the last character moved. The EOS flag is set if the ETX in the destination string would have been overstored and transfer stops with the character that would have overstored the ETX.

The MOVE instruction can also move character strings to numeric strings and vice versa. (The movement of numeric strings to numeric strings is discussed in section 3.4.5.) A character string will be moved to a numeric string only if the character string is of valid numeric format (only digits, spaces, a leading minus sign, and one decimal point allowed). Otherwise, the numeric string is set to zero. Note that only the part of the character string starting with the formpointer is considered in the validity check and transferred if the string is of valid numeric format. The number in the character string will be reformatted to conform to the format of the numeric string. The TYPE instruction (see Section 3.3.10) is available to allow checking the character string for valid numeric format before using the MOVE instruction. When a numeric string is moved to a character string, all characters of the numeric item (unless the ETX would be overstored) are transferred starting with the first physical character in the destination string. The formpointer of the destination string is set to one and the logical length is set to point to the last character transferred.

Examples:

```
MOVE STRING TO STRING
MOVE A,B
MOVE STRING TO NUMBER
MOVE NUMBER,STRING
```

Since Databus 3 has no facility for handling numeric variables, they allow moving strings to strings, strings to numeric indexes, and vice versa (see Section 3.5.4 for the details). This makes it possible for the Interpreters which have these instructions to PRINT, DISPLAY, and WRITE index values, as well as initialize indexes to values input from the keyboard or read from tape.

3.3.5 APPEND (2)(3)

APPEND appends the source string to the destination string. The characters appended are those from under the formpointer through under the logical length pointer of the source string. The characters are appended to the destination string starting <u>after the formpointed character</u> in the destination string. The source string pointers remain unchanged, but the destination string pointers both point to the last character transferred. The EOS condition will be set if the new string will not fit physically into the destination string, but all characters that will fit will be transferred.

Examples:

```
APPEND SOURCE TO DEST
APPEND NAME,BUFF
```

3.3.6 RESET (2)(3)

There are two versions of the RESET command in CTOS Databus. One version works with version 3 interpreters; the other works with version 4 interpreters.

Version 3 Interpreters:

RESET changes the value of the formpointer of the source string to the value indicated by the second operand. If no second operand is given, the formpointer will be reset to one. The second operand must be a positive number less than 128. The EOS condition will be set and no change will occur if the requested position is greater than the string's logical length.

Version 4 Interpreters:

RESET changes the value of the formpointer of the source string to the value indicated by the second operand. If no second operand is given, the formpointer will be reset to one. The second operand may be a quoted character, in which case the ASCII value minus 32 (space gives zero, ! one, " two, etc.) will be used for the value of the formpointer of the source string. The second operand may also be a character string, in which case the ASCII value minus 32 of the character under the formpointer of that string will be used for the value of the formpointer of the source string. The second operand may also be a numeric string or a number, in which case the value of the number will be used for the formpointer of the source string.

RESET also has the capability of extending the logical length of the first operand. If the formpointer value specified is past the logical length of the first operand, the logical length will be extended until it will accommodate the formpointer value. If this would cause the logical length to be past the physical end of the string, the logical length and formpointer will both be left pointing to the last physical character in the string. This feature is useful in extracting and inserting information within a large string. The EOS condition will be set if a change in the logical length of the first operand occurs.

Examples:

```
RESET XDATA TO 5
RESET Y
RESET Z TO NUMBER
RESET Z TO STRING
```

Note that the RESET instruction is very useful in code conversions and hashing of character string values as well as large string manipulation.

3.3.7 BUMP (2)(3)

There are two versions of the BUMP command in CTOS Databus. One version works with version 3 interpreters; the other works with version 4 interpreters.

Version 3 Interpreters:

BUMP increments the formpointer if the result will be within the string (between 1 and the logical length). An EOS condition will occur if the formpointer is equal to or greater than the length and it will not be incremented.

Version 4 Interpreters:

BUMP increments or decrements the formpointer if the result will be within the string (between 1 and the logical length). If no parameter is supplied, BUMP increments the formpointer by one. However, a positive or negative literal value may be supplied to cause the formpointer to be moved in either direction by any amount. An EOS condition will be set and no change in the formpointer occurs if it would be less than one or greater than the logical length after the movement had occurred.

Examples:

```
BUMP CAT
BUMP CAT BY 2
BUMP CAT,-1
```

3.3.8 ENDSET (2)(3)

ENDSET causes the operand's formpointer to point where its logical length points.

Example:

```
ENDSET PNAME
```

3.3.9 LENSET (2)

The LENSET command is implemented in Version 4 Interpreters only. LENSET causes the operand's logical length to point where its formpointer points.

Example:

```
LENSET QNAME
```

3.3.10 TYPE (2)

TYPE sets the EQUAL and ZERO condition if the string is of valid numeric format (only leading minus, one decimal point, and digits or spaces).

Example:

```
TYPE ALPHA
```

3.3.11 EXTEND (2)(3)

EXTEND increments the formpointer, stores a space in the position under the new formpointer, and sets the logical length to point where the new formpointer points if the new logical length would not point to the ETX at the end of the character string. Otherwise, the EOS flag is set and no other action is taken.

Example:

```
EXTEND BUFF
```

3.3.12 CLEAR (2)(3)

CLEAR causes the operand's logical length and formpointer to be zero.

Example:

```
CLEAR NBUFF
```

3.3.13 RANGE (4)(5)

The RANGE instruction sets the EQUAL condition code if the operand one string characters are within the limits specified by the second and third operands. Operands two and three can be quoted alphanumerics or numeric values less than 256. The RANGE instruction compares each character of the string variable to see that each is greater than or equal to operand two and less than or equal to operand three. This instruction is particularly useful to determine whether a string is alphabetic or numeric.

Examples:

```
     RANGE   XDATA,"0","9"
     RANGE   YDATA,"A","Z"
     RANGE   YDATA, 0101,0132
```

3.3.14 LOAD (2)(3)

LOAD performs a MOVE from the character string pointed to by the index numeric operand, the second operand, to the first character string specified. In the Databus Interpreters which have numeric string variables, the index is a numeric string variable. In the Databus Interpreters which do not have numeric string variables, the numeric indexes I0 through I7, which have been set up in these interpreters may be used. The instruction has no effect if the index is negative, zero, or greater than the number of items in the list. Note, that the index is truncated to no decimal places before it is used (e.g. 1.7=1).

For example:

```
     LOAD AVAR FROM N OF NAME,TITLE,HEDING
```

causes the contents of one of the variables in the list, based on the value of the numeric variable N, to be moved into the first operand AVAR.

```
     LOAD ANS FROM I2 OF VENDOR,ACCT,QUAN,ITEM
```

causes the contents of one of the variables in the list, based on the value of the numeric index I2, to be moved to the first operand ANS.

3.3.15 STORE (2)(3)

STORE performs a MOVE from the first character string specified to a character string in a list specified by an index numeric operand given as the second operand. In the Databus Interpreters which have numeric string variables, the index is a numeric variable. In the Databus Interpreters which do not have numeric string variables, the numeric indexes I0 through I7, which have been set up in these interpreters may be used.

The instruction has no effect if the index is negative, zero, or greater than the number of items in the list. Note that the index is truncated to no decimal places before it is used (e.g. 1.7=1).

For example:

```
STORE Y INTO NUM OF ITEM,ENTRY,ALINK,LIST
```

causes the contents of the first operand Y to be moved into one of the variables in the list, based on the value of the numeric variable NUM.

```
STORE VAR INTO I3 OF STR1,STR2,STR3,STR4
```

causes the contents of the first operand VAR to be moved into one of the variables in the list, based on the value of the numeric index I3.

The LOAD and STORE instruction statements may be continued to the next line if a colon (:) is the terminating character of the instruction. The colon replaces the comma separating the last entry of the first line from the first entry of the second line. The first entry of the second line should begin in the instruction field.

Examples:

```
LABEL LOAD SYMBOL FROM N OF VAR,CONST,DEC:
      CNT,FLAG,LIST
NEXT  STORE NAME INTO I0 OF A,B,C,D,E,F,G:
      H,I,J,K,L,M
```

3.4 Numeric String Variable Arithmetic Instructions

All of the numeric variable arithmetic instructions have certain characteristics in common. Except for LOAD and STORE, each numeric variable arithmetic instruction is always followed by two numeric string variable names. The contents of the first variable is never modified and, except in the COMPARE instruction, the contents of the second variable always contains the result of the operation.

For example in:

```
     ADD XAMT TO YAMT
```

the content of XAMT is not changed, but YAMT contains the sum of XAMT and YAMT after the instruction is executed.

Following each numeric string variable arithmetic instruction, the condition flags, OVER, LESS, and ZERO (EQUAL) are set to indicate the results of the operation. OVER indicates that the result of an operation is too large to fit in the space allocated for the variable (a result is still given with truncation to the left and rounding to the right, however). LESS indicates that the content of the second variable is negative following the execution of the instruction (or would have been in the case of COMPARE). ZERO (EQUAL) indicates that the value of the second variable is zero following the execution of the instruction.

Whenever overflow occurs, the higher valued digits that do not fit the variable are lost. For example, a variable is defined:

```
     NBR42   FORM 2.2
```

and a result of 4234.67 is generated for that variable, NBR42 will contain only 34.67.

Whenever an operation produces lower order digits than a variable was defined for, the result is rounded up. A variable with the FORM 3.1 would contain:

```
      46.2 for  46.213
     812.5 for 812.483
       3.7 for   3.666
       3.9 for   3.850
```

Note that if an OVER occurs during an ADD, SUB, or COMPARE of two strings of different physical lengths, the result and the LESS condition flag may not be correct.

3.4.1 ADD (1)(2)

ADD causes the content of variable one to be added to the content of variable two.

Examples:

```
ADD X TO Y
ADD DOG,CAT
```

3.4.2 SUB (1)(2)

SUB causes the content of variable one to be subtracted from the content of variable two.

Examples:

```
SUB RX350 FROM TOTAL
SUB Z,TOTAL
```

3.4.3 MULT (1)(2)

MULT causes the content of variable two to be multiplied by the content of variable one.

Examples:

```
MULT DICK BY HARRY
MULT W,Z
```

3.4.4 DIV (1)(2)

DIV causes the content of variable two to be divided by the content of variable one. The number of decimal places in the result is equal to the number of decimal places in variable two minus the number of decimal places in variable one. And the number of places to the left of the decimal point in the result is equal to the number of places to the left of the decimal point in the variable two minus the number of places to the left of the decimal point in variable one. If the number is negative, it is assumed to be zero. For example, if a number that is defined by FORM 3.2 is divided into a number defined as FORM 6.5, the result will be a number of FORM 3.3. Therefore, a user should be very careful in defining numeric variables to be used in divide operations.

Examples:

```
DIV SFACT INTO XRSLT
DIV X3,HOURS
```

3.4.5 MOVE (1)(2)

MOVE causes the content of variable one to replace the content of variable two.

Examples:

```
MOVE FIRST TO SECOND
MOVE A,B
```

3.4.6 COMPARE (1)(2)

COMPARE does not change the content of either variable but sets the condition flags exactly as if a SUB instruction had occurred.

Examples:

```
COMPARE XFRM TO YFRM
COMPARE RING,DING
```

Care should be used in defining variables to be compared. Comparison of variables in which the length of the first variable is longer than the length of the second variable results in an overflow condition. The OVER flag is set, and the EQUAL or ZERO flag is set to show the result of the comparison. However, the LESS flag is not set in this case.

3.4.7 LOAD (1)(2)

The LOAD instruction for numeric string variables selects an operand out of the list based on the index operand. It then performs a MOVE operation from the contents of the selected variable into the first operand. In the Databus Interpreters which have numeric variables the index is a numeric variable. In the Databus Interpreters which do not have numeric variables, the numeric indexes I0 through I7, which have been set up in the Interpreters may be used. If the index is negative, zero, or greater than the number of items in the list, then the instruction has no effect, note that the index is rounded to the nearest integer before it is used (e.g. 1.7=1).

For example:

```
LOAD CAT FROM N OF FACT,MULT,SPACE
```

causes the contents of one of the variables in the list, based on the value of the numeric variable N to be moved into the first operand CAT.

```
          LOAD SUM FROM I6 OF TOTAL,SUBTOT,PROD,DIFF
```

causes the contents of one of the variables in the list, based on the value of the numeric index I6, to be moved to the first operand SUM.

3.4.8 STORE (1)(2)

The STORE instruction for numeric variables selects an operand out of the list based on the index operand. It then performs a MOVE operation from the contents of the first operand into the selected variable. In the Databus Interpreters which have numeric variables, the index is a numeric variable. In the Databus Interpreters which do not have numeric variables, the numeric indexes, I0 through I7, which have been set up in these Interpreters may be used. if the index is negative, zero, or greater than the number of items in the list, the instruction has no effect. Note that the index is rounded to the nearest integer before it is used (e.g. 1.7 =1).

For example:

```
          STORE X INTO NUM OF VAL,SUB,TOT
```

causes the contents of the first operand X to be moved into one of the variables in the list, based on the value of the numeric variable NUM.

```
          STORE RES INTO I5 OF DIV,MUL,ADD,SUB
```

causes the contents of the first operand RES to be moved into one of the variables in the list, based on the value of the numeric index I5.

The LOAD and STORE instruction statements may be continued to the next line if a colon (:) is the terminating character of the instruction. The colon replaces the comma separating the last entry of the first line from the first entry on the second line. The first entry of the second line should begin in the instruction field.

Examples:

```
          LABEL LOAD NUMBER FROM N OF N1,N2,N3,N4,N5:
              N6,N7,N8,N9
          ENTRY STORE COUNT INTO I2 OF TIME,RATE,DIST,SPG:
              COST,TOT,SUM
```

3.5 Numeric Index Arithmetic Instructions

In the Databus Interpreters which do not have numeric variables, some numeric indexes have been set up. These indexes do not need to be set up in the user's program. There are eight indexes which are referred to as I0 through I7. They are initialized to zero at the beginning of every program.

These indexes have been set up to be used as counters. They may be any integer value between 0 and 127 decimal.

Each numeric index arithmetic instruction is followed by two operands. The first may be an index or a number. This number may be an octal or decimal (octal if it is preceded by a 0 (e.g., 017) number between 0 and 127 decimal. The second operand must be one of the indexes. The content of the first operand is never modified, and except in the COMPARE instruction, the contents of the second variable always contain the result of the operation.

For example in:

```
ADD I1 TO I3
```

the content of I1 is not changed, but I3 contains the sum of I1 and I3 after the instruction is executed.

Following each arithmetic instruction, three "flags" are set within the processor to indicate the results of the operation. These flags are LESS, EQUAL, and ZERO. LESS indicates that the content of the second variable is negative following the execution of the instruction. EQUAL and ZERO indicate that the value of the second variable is zero following the execution of the instruction.

The preposition connecting the two variables can be replaced with a comma as a shorter means of writing the statement.

For example:

```
          SUB I1 FROM I3
     and
          SUB I1,I3
```

are equivalent. Note that a space cannot separate the first variable and the comma or an E-flag will occur during compilation.

3.5.1 ADD (3)(4)(5)

ADD causes the content of operand one to be added to the content of operand two.

Examples:

```
ADD I1,I5
ADD 1 TO I1
ADD 1,I3
```

3.5.2 SUB (3)(4)(5)

SUB causes the content of operand one to be subtracted from the content of operand two.

Examples:

```
SUB I1 FROM I2
SUB 1,I7
SUB 10 FROM I3
```

3.5.3 COMPARE (3)(4)(5)

COMPARE does not change the content of either operand but sets the condition flags exactly as if a SUB instruction had occurred.

Examples:

```
COMPARE I3 TO I4
COMPARE I0,I4
COMPARE 35 TO I1
```

3.5.4 MOVE (3)(5)

The Index MOVE instruction allows the user to move indexes to strings and strings to indexes. This makes it possible for the interpreters which have this instruction to PRINT, DISPLAY, and WRITE index values, as well as initialize indexes to values input from the keyboard and read from tape.

Examples:

```
MOVE I1 TO STRING
```

moves the index I1 to the string variable string. The index move to a string sets the logical length of the string to 3 if the physical length is greater than or equal to 3. If the physical length is less than 3, as much of the index value is moved as the string will hold starting at the right most digit of the index. The formpointer is set to 1. If the value of the index is 3, a move to a string of length 3 or more will result in the string 003. The resulting string value will always be decimal.

```
          MOVE STRING,I3
```

moves the string variable STRING to the index I3. The string value is assumed to be decimal. The string value to be moved should not be more than 127 decimal.

Note that the name of any variable that is to be used in an index move instruction should not begin with the letter I, so as not to be confused with the indexes I0-I7 by the compiler.

3.6 KEYBOARD,C.R.T.,PRINTER INPUT/OUTPUT Instructions

These statements move data between the program variables and the keyboard, screen, or printer. They each allow a list of variables to follow the operation mneumonic. This list may be continued on more than one line with the use of the colon. The I/O list may contain some special control information besides the names of the variables to be dealt with. DATABUS has no formatting information other than the list controls and that which is implied by the format of the variables. The number of characters transferred is always equal to the number of characters physically allocated for the string, therefore, allowing the programmer to set up his formatting the way he dimensions his data variables.

3.6.1 KEYIN (1)(2)(3)(4)(5)

KEYIN causes data to be entered into either character or numeric strings from the keyboard. A single KEYIN instruction may contain many variable names and list control items. When characters are being accepted from the keyboard, the flashing cursor is on. At all other times, the cursor is off.

When a numeric variable is encountered in a KEYIN statement, only an item of a format acceptable to the variable (not too many digits to the left or right of the decimal point and no more than one sign or decimal point) is accepted. If a character is struck that is not acceptable to the format of the numeric variable, the character is ignored and the Datapoint 2200 signals a "beep". Note that if fewer than the allowable number of digits to the left or right of the decimal point are entered, the number entered will be reformatted to match the format of the variable being stored into. When the ENTER key is struck, the next item in the instruction list is processed.

When a character string variable is encountered, the system accepts any set of ASCII characters up to the limit of the physical length of the string. The formpointer of the string variable is set to one, and characters are stored consecutively starting at the physical beginning of the string. When the ENTER key is struck, the logical length is set to the last character entered, and the next item in the keyin list is processed.

Other than variable names, the KEYIN instruction may contain quoted items and list controls. Quoted items are simply displayed as they are shown in the statement. The list controls begin with an asterisk and allow such functions as cursor positioning and screen erasure. The *H<n> control causes the cursor to be positioned horizontally to the position specified by n. The *V<n>

control causes the cursor to be positioned vertically to the position specified by n. Note that these numbers are literals. The horizontal position is restricted by the interpreter to be from 1 to 80 and the vertical position is restricted to be from 1 to 12. The *EF control erases the screen from the current cursor position, the *EL control erases the rest of the line from the current cursor position, and the *R control causes the screen to be rolled up one line.

The KEYIN and DISPLAY instructions in Version 4 Interpreters have been expanded to allow *C and *L list controls. The *C control causes the cursor to be set to the beginning of the current line, and the *L control causes the cursor to be set to the following line in the current horizontal position. The *H<n> and *V<n> controls have also been changed so that the numbers specified by n may be literals or numeric variables. Numbers outside of the horizontal or vertical position ranges have the effective value of 1.

Normally, the cursor is positioned to the start of the next line at the termination of a KEYIN statement. However, placement of a semicolon after the last item in the list will cause this positioning to be suppressed, allowing the line to be continued with the KEYIN or DISPLAY statement. This feature is also true of the PRINT command.

Examples:

```
KEYIN *H1,*V1,*EF,"NAME: ",NAME,*H35,*V2,"ACNT NR: ":
      ACTNR, " ADDRESS: ",STREET,*H10,*V3,CITY:
      *HX,*V4,"ZIP: ",ZIP;
```

While keying a given variable, the operator may strike the BACKSPACE key and cause the last character entered to be deleted. He may also strike the CANCEL key and cause all of the characters entered for that variable to be deleted. Whenever an input from the keyboard is expected, the cursor flashes on and off. It remains off at all other times.

3.6.2 DISPLAY (1)(2)(3)(4)(5)

DISPLAY follows the same rules as the KEYIN except that when a variable name is encountered in the list following the instruction, the variable's contents are displayed instead of keyed in.

In the old tape format interpreters, DISPLAY begins displaying at the formpointed character of string variables and continues through the logical length.

In the new tape format interpreters, character strings are displayed starting with the first physical character and continuing through the logical length. Spaces will be displayed for any character positions that exist between the logical length and physical end of the string. Numeric strings are always displayed in their entirety in both interpreters.

Examples:

```
 DISPLAY *H5,*V1,"RATE: ",RATE:
         *H5,*V2,"AMOUNT: ",AMNT
```

3.6.3 PRINT (1)(2)(3)(4)(5)

The PRINT instruction causes the contents of variables in the list to be printed in a fashion similar to the way DISPLAY causes the contents of variables to be displayed. The list controls are much the same as DISPLAY except that cursor positioning cannot be used, column tabulation is provided: *<n> causes tabulation to column <n> unless that column has been passed (however, for Servo Printer backward tabulation is allowed), *F causes an advance to the top of the next form, *L causes a line feed to be printed, and *C causes a carriage return to be printed. The PRINT statement may be continued on more than one line by use of the colon.

In the old tape format interpreters, PRINT begins printing at the formpointed character of string variables and continues through the logical length of the string.

In the new tape format interpreters, PRINT begins printing at the first character of the string and continues through the physical end of the string. Blanks are printed for all characters after the logical end of the string. Numeric variables are printed in their entirety in both interpreters.

Examples:

```
     PRINT *20,"TRANSACTION SUMMARY",*C,*L:
           PNAME,*C,*L,*10,RATE,*20,HOURS,*30:
           AMNT,*L
```

3.6.4 BEEP (1)(2)(3)(4)(5)

The BEEP instruction causes the machine to produce an audible tone.

Example:

```
    BEEP
```

3.6.5 CLICK (1)(2)(3)(4)(5)

The CLICK instruction causes the machine to produce an audible click.

Example:

```
CLICK
```

3.6.6 DSENSE (1)(2)(3)(4)(5)

The DSENSE instruction tests the DISPLAY key sense switch. If the DISPLAY key has been depressed, then the EQUAL condition flag is set. If the DISPLAY key is not depressed then the EQUAL condition flag is reset.

Example:

```
DSENSE
```

3.6.7 KSENSE (1)(2)(3)(4)(5)

The KSENSE instruction acts like DSENSE except that it tests the KEYBOARD key sense switch.

Example:

```
KSENSE
```

3.7 Cassette Tape I/O Instructions

3.7.1 READ (1)(2)(3)(4)(5)

The READ command causes a record to be read from the indicated tape deck and the data entered into the variables appearing in the list following the READ instruction.

For _old tape format_ READ instructions the following is true: As the data is entered into the variables, the formpointer of each string variable is set to one and the characters are stored consecutively in the strings starting at the beginning of the string. The logical length is the same as the physical length of the variable on the tape. If the record contains more items than the list, the remaining unused variables will be disregarded. If the list contains more variables than were in the record, a format trap occurs. If any variable from the record contains more characters than the physical length of the list variable will hold, a FORM trap is set. A FORM trap also occurs if the data read in is of different type than the variables in the list.

The new tape format records no longer contain the length, formpointer, 0200, or ETX of variables. Only the actual data characters are written. Since there are no delimiters between variables, the entire physical length of strings starting at the first character is written to tape. Blanks are written for all characters after the logical end of the string. When the record is read the data is entered into the variables starting at the first position in the string and continuing to the physical end. The formpointer is set to one and the logical length is set to the length of the string at the last non-blank character. If the record contains more items or characters than were in the record, the extra strings are blank filled, and the numbers are zeroed. If the variables in the READ instruction are not the same size as the variables in the WRITE instruction for that record, some of the characters may be stored into the wrong variables. However, this may be useful to reformat variables when they are read.

The only error condition given in the new tape format is a read failure trap, RFAIL. This will occur if the record read is more than 249 data characters long, or if a string is read into a numeric variable.

The number 1 or 2 must appear as the first item in the READ instruction list to indicate which deck is to be read (rear or front respectively).

Examples:

```
READ 1,A,B,TOTAL
READ 2,NAME,ADR,AIP
```

If a WRITE instruction has occurred to the indicated deck without a WEOF instruction (the deck is in write mode), the READ instruction will abort the program in the old tape format interpreters. In the new tape format this is allowed.

NOTE: It is not necessary to always read every variable from a record. For example, records of five variables each were written to tape using the following write instruction.

```
WRITE 1,NAME,COMPANY,ADDRESS,SSN,POSITION
```

Another program might use the same tape, but only need the company name from each record. So this program could use the following instruction.

```
READ 1,NAME,COMPANY
```

Every variable up to and including the variables desired must be in the read statement in the order the variables appear in the records on tape. Each read instruction issued, advances the tape one record. To advance the tape past a record, only the instruction

```
        READ 1
    or
        READ 2
```

is needed. This is particularly useful for positioning a tape to the end of file.

3.7.2 WRITE (1)(2)(3)(4)(5)

The WRITE instruction causes a record to be written to the indicated deck. The record will contain the variables indicated in the list following the WRITE instruction.

For old tape format tapes, the record may be any length up to 240 characters. Each numeric variable will have a length equal to its defined length plus 2 and each character string will have a length equal to its logical length plus 3. WRITE begins writing at the formpointed character of string variables and continues to the logical end of the string. An attempt to write more than 240 characters will abort the program.

For new tape format tapes, the record may be any length up to 249 data characters. Since only the actual data characters are written to tape, each numeric and character string variable will have a length equal to its defined physical length. WRITE begins writing at the first character of string variables and continues to the physical end of the string. Blanks are written for all characters after the logical end of the string. Using this technique, a WRITE statement will always write the same number of characters for a variable, no matter what the logical length of the string variable.

The number 1 or 2 must appear as the first item in the WRITE instruction list to indicate which deck is to be written to (1 indicates the rear deck, 2 the front deck).

In the old tape format interpreters, once a WRITE instruction is issued to a given deck, it is in write mode and no other instructions can be issued to that deck except WRITE and WEOF. Once WEOF is issued, it is in the read mode and any instruction may be issued.

Examples:

```
WRITE 2,TIME,TOTAL,NAME
WRITE 1,FORM1,FORM2,FORM3
```

The READ and WRITE instruction statements may be continued to the next line if a colon (:) is the terminating character of the instruction. The colon replaces the comma separating the last entry of the first line from the first entry on the second line. The first entry of the second line should begin in the instruction field.

Examples:

```
START READ 1,NAME,POSN,ADDR,SSN,INS:
      CODE,ITEM,QUANT
WR    WRITE 2,NAME,POSN,ADDR,SSN,INS:
      CODE,ITEM,QUANT
```

3.7.3 REWIND (1)(2)(3)(4)(5)

The REWIND instruction list contains only a 1 or 2 to indicate the rear or front deck respectively. If the rear deck is indicated, the tape will slew to the beginning of the file area following the program library on the rear cassette. If the front deck is indicated, the cassette will be high-speed rewound to the beginning of the tape and the head positioned to the beginning of the first data record.

The REWIND instruction will abort the program if there has been a WRITE instruction to the deck without a following WEOF instruction.

NOTE

> A PREPARE or REWIND instruction must be issued to deck 1 before any other tape instruction can be issued to that deck. A REWIND instruction is not necessary for deck 2, but is usually desirable. However, if two or more programs are being chained, the user may wish to have each new program continue writing to deck 2 where the previous program left off. In this case a REWIND instruction would not be desired for deck 2. Note, however, that a WEOF must be issued before the chain is performed in the old tape format interpreters. This will not result in an error in the new tape format interpreters, but is usually desirable.

Example:

```
REWIND 1
```

3.7.4 BKSP (1)(2)(4)(5)

The BKSP instruction causes the indicated deck to backspace one record. If the tape is at the beginninq of the file no backspace occurs and an EOF trap occurs.

A 1 or 2 must follow the BKSP instruction to indicate the rear or front deck respectively.

If the indicated deck is in write mode, a BKSP will cause the program to abort.

Example:

```
BKSP 2
```

3.7.5 PREPARE (1)(2)(3)(4)(5)

The PREPARE instruction list contains only a 1 or 2 to indicate the rear or front deck respectively. If the rear deck is indicated, the instruction performs the same function as REWIND. If the front deck is indicated, the cassette is rewound and a new beginning-of-file marker is written.

Example:

```
PREPARE 2
```

3.7.6 WEOF (1)(2)(3)(4)(5)

The WEOF instruction causes an end-of-file mark to be written on the indicated deck and causes that deck to be taken out of write mode. The tape is left positioned before the file marker.

A 1 or 2 must follow the WEOF instruction to indicate the rear or front deck respectively.

Example:

```
WEOF 1
```

3.7.7 BSPR (3)

The BSPR instruction is the same as the BKSP instruction. Databus 3 uses this command instead of BKSP to differentiate the backspace record and backspace file instructions.

Example:

```
BSPR 1
```

3.7.8 BSPF (3)

The BSPF instruction causes the indicated cassette deck to backspace one file. Since only one file is allowed on a DATABUS Cassette the BSPF instruction for cassette performs the same function as the REWIND command (see Section 3.7.3).

A 1 or 2 must follow the BSPF to indicate the rear or front deck respectively.

In the old tape format interpreters, if the indicated deck is in write mode, a BSPF will cause the program to abort.

Example:

```
BSPF 2
```

3.7.9 ADVR (3)

The ADVR instruction causes the indicated deck to advance the tape one record. If the tape is at the end of the file no advance occurs and an EOF trap occurs.

A 1 or 2 must follow the ADVR instruction to indicate the rear or front deck respectively. If the indicated deck is in write mode, an ADVR will cause the program to abort.

Example:

```
ADVR 1
```

3.7.10 ADVF (3)

The ADVF instruction causes the indicated cassette deck to advance the tape to the end of file. The tape is positioned to the end of file 32 on the rear deck and file 0 on the front deck.

A 1 or 2 must follow the ADVF instruction to indicate the rear or front deck respectively.

If the indicated deck is in write mode an ADVF will cause the program to abort.

Example:

```
ADVF 2
```

3.8 Industry Compatible Magnetic Tape I/O Instructions

Either 7-Track or 9-Track Tapes may now be written with Databus 3. On 9-Track, either ASCII, EBCDIC, or BCD may be used. On 7-Track, only BCD may be used. There are two versions of the Databus 3 Interpreter. One contains the EBCDIC tables, the other contains the BCD tables. See Section 7 for Tape formats.

The tape records differ from cassette records in that only the actual data characters are written to tape. The length, formpointer, and ETX (0203) which appear in cassette records are not written in mag tape records. Since there are no delimiters between string variables, the characters from the formpointed character through the physical end of each variable are written to tape. All characters after the logical end are written as blanks. When the variables are read back from tape, the length of each string is set to its physical length, because the tape READ stores characters in a variable starting at the beginning of the string and continuing up to the physical end of the string.

The industry compatible magnetic tape files differ from cassettes in that there may be many files on one tape. The files are separated by a single EOF tape mark with two EOF tape marks indicating the end of data on a tape.

3.8.1 READ (3)

The READ command causes a record to be read from the indicated tape deck and the data entered into variables appearing in the list following the READ instruction. As the data is entered into the variables, the formpointer of each string variable is set to one and the characters are stored consecutively into the strings starting at the beginning of the string. The length is set to the physical length of the string.

If the record contains more items than the list, the remaining unused variables will be disregarded. If the list contains more variables than were in a record, an RFAIL trap occurs. If the total number of characters in the record is greater than the total number of characters that may be stored in the string variables in the list, an RFAIL trap is set. If the variables in the READ instruction for a record are not the same size as the variables in the WRITE instruction for that same record, some of the characters may be stored into the wrong variables.

The number 3 or 4 must appear as the first item in a tape READ instruction list to indicate which tape unit is to be used. (3=adr 264, 4=adr 113) If only one tape is used in a configuration, 3 should be the correct tape address.

Examples:

```
READ 3,SUM,PROD,DIFF
READ 4,SSN,COMP,VAR1,VAR2
```

If a WRITE instruction has occurred to the indicated deck without a WEOF instruction (the deck in write mode), the READ instruction will abort the program.

NOTE: It is not necessary to read every variable from a record. For example, records of five variables each were written to tape with the following instruction:

```
WRITE 3,NAME,COMPANY,ADDRESS,SSN,POSITN
```

Another program might use the same tape, but only need the company name from each record. So this program could use the following instruction:

```
READ 3,NAME,COMPANY
```

Every variable up to and including the variables desired must be in the read statement in the order the variables appear in the records on tape. Each read instruction issued advances the tape one record.

3.8.2 WRITE (3)

The WRITE instruction causes a record to be written to the indicated tape deck. The record will contain the variables indicated in the list following the WRITE instruction. The record may be any length up to 1057 characters. The characters from the formpointed character through the physical length (up to the ETX (0203)) are written to tape. An attempt to write more than 1057 characters will abort the program.

The number 3 or 4 must appear as the first item in the WRITE instruction list to indicate which tape unit is to be written to (3=adr 264, 4=adr 113). Users with only one tape in their configuration should use 3 for the correct tape address.

Once a WRITE instruction is issued to a given deck, it is in write mode and no other instructions can be issued to that deck except WRITE and WEOF. Once WEOF is issued, the deck is in read mode and any instruction may be issued.

Examples:

```
WRITE 3,TIME,TOTAL,NAME
WRITE 4,CODE,INS,REF,MODEL,MAKE
```

The READ and WRITE instructions statements for magnetic tape may be continued to the next line if a colon (:) is the terminating character of the instruction. See cassette READ and WRITE for examples.

3.8.3 REWIND (3)

The REWIND instruction list for tape contains a 3 or 4 to indicate which tape unit to address (3=adr 264, 4=adr 113). Once the correct unit is addressed, the tape is rewound to the beginning. No positioning is necessary because the first record on tape is data.

NOTE

> A PREPARE and REWIND instruction issued to the industry compatible magnetic tape unit is usually desirable before any other tape instruction is issued to that deck. However, if two or more programs are being chained, the user may wish to have each new program continue writing to the tape where the previous program left off. In this case, a REWIND would not be desired. Note, however, that a WEOF must be issued before the chain is performed.

Example:

```
REWIND 3
```

3.8.4 PREPARE (3)

The PREPARE instruction list for tape contains only a 3 or 4 to indicate which tape unit to address (3=adr 264, 4=adr 113). Once the correct unit is addressed, the tape is rewound and an end-of-file is written on the tape. The end-of-file mark consists of two EOF tape marks. The tape is then backspaced over the two file marks just written.

Example:

```
PREPARE 4
```

3.8.5 WEOF (3)

The WEOF instruction causes two end-of-file tape marks to be written to the tape. The tape is then backspaced over the two file marks and left positioned before the first file mark. The indicated tape unit is taken out of write mode.

A 3 or 4 must follow the WEOF instruction to indicate the tape unit to be addressed (3=adr 264, 4=adr 113).

Example:

```
    WEOF 3
```

3.8.6 BSPR (3)

The BSPR instruction causes the indicated tape unit to backspace the tape one record. If the tape is at the beginning of a file no backspace occurs and an EOF trap is set. If the tape is at the beginning of tape no backspace occurs and the EOT trap is set. If the backspace moves the tape to the beginning of tape, the EOT trap also is set.

A 3 or 4 must follow the BSPR to indicate which tape deck to address (3=adr 264, 4=adr 113).

If the indicated drive is in write mode, a BSPR will cause the program to abort.

Example:

```
    BSPR 3
```

3.8.7 BSPF (3)

The BSPF instruction causes the indicated tape unit to backspace the tape one file. If the indicated drive is at the beginning of tape, no backspace occurs, but no traps are set. When the backspace does occur, the tape is left positioned at the beginning of the previous file.

A 3 or 4 must follow the BSPF instruction to indicate which tape unit is to be addressed (3=adr 264, 4=adr 113).

If the indicated drive is in write mode, a BSPF will cause the program to abort.

Example:

```
    BSPF 4
```

3.8.8 ADVR (3)

The ADVR instruction causes the indicated tape to advance the tape one record. If the tape is at the end of a file no advance occurs and an EOF trap is set. If the tape is at the end of tape no advance occurs and the EOT trap is set. If the advance moves the tape to the end of tape the EOT trap also is set.

A 3 or 4 must follow the ADVR instruction to indicate which tape unit to address (3=adr 264, 4=adr 113).

If the indicated deck is in write mode, an ADVR will cause the program to abort.

Example:

```
ADVR 4
```

3.8.9 ADVF (3)

The ADVF instruction causes the indicated tape to advance the tape one file. If the tape is at the end of the last file no advance occurs and an EOF trap occurs. If the tape is at the end of tape the EOT trap is set. If the advance moves the tape to the end of tape the EOT trap also is set. If the ADVF occurs the tape is left at the beginning of the following file.

A 3 or 4 must follow the ADVF instruction to indicate which tape deck to address (3=adr 264, 4=adr 113).

If the indicated deck is in write mode, an ADVF will cause the program to abort.

Example:

```
ADVF 3
```

3.8.10 ADVFW (3)

The ADVFW instruction causes the indicated tape deck to advance the tape past the next tape mark so that a new file may be written. The indicated deck is then put into write mode. If the tape is at the end of tape, or if the advance moves the tape to the end of tape an EOT trap occurs.

A 3 or 4 must follow the ADVFW instruction to indicate which tape unit to address (3=adr 264, 4=adr 113).

If the indicated deck is in write mode, an ADVFW will cause the program to abort.

Example:

```
ADVFW 4
```

3.8.11 PBOF (3)

The PBOF instruction causes the indicated deck to be positioned to the beginning of the file in which the tape is currently positioned.

A 3 or 4 must follow the PBOF instruction to indicate which tape unit to address (3=addr 264, 4=adr 113).

If the indicated deck is in write mode, a PBOF will cause the program to abort.

Example:

```
PBOF 3
```

3.8.12 PEOF (3)

The PEOF instruction causes the indicated drive to be positioned to the end of the file in which the tape is currently positioned.

A 3 or 4 must follow the PEOF instruction to indicate which tape unit to address (3=adr 264, 4=adr 113).

If the indicated drive is in write mode, a PEOF will cause the program to abort.

Example:

```
PEOF 4
```

3.8.13 ASCII (3)

The ASCII instruction places the industry compatible magnetic tape in a mode which will read and write ASCII tapes. If no tape mode instruction is given, the tape is assumed to be ASCII.

Example:

```
ASCII
```

3.8.14 EDCDIC (3)

The EBCDIC instruction places the industry compatible magnetic tape in a mode which will read and write EBCDIC tapes. If no tape mode instruction is given, the tape is assumed to be ASCII. This command is for 9-Track tape units only.

Example:

```
EBCDIC
```

3.8.15 BCD (3)

The BCD command places the industry compatible magnetic tape in a mode which will read and write BCD tapes. If no tape mode instruction is given, the tape is assumed to be ASCII. This command should be used when writing to 7-Track tape units.

Example:

```
     BCD
```

3.9 Communications I/O Instructions

3.9.1 SEND (3)

The SEND instruction causes data to be transmitted from one 2200 to another over a data line through a 202 internal modem at 1200 baud. The data sent is from the list of items following the SEND instruction. The list items may be either string variables or quoted character strings. There is no limit to the number of characters that may be sent.

Example:

```
     SEND NAME,ADDR,SSN
     SEND "ACK"
     SEND "NAME",NAME,"POSITION",POSN
```

The message sent is of the following format:

```
     RO/RO/RO/STX/string/015/string/015/---/ETX/LRC/RO/RO

     RO=Rubout                ETX=End of Message
     STX=Start of Message     LRC=Longitudinal Record Parity
```

A string, in the above example, can be either a string variable or a quoted character string. Each string variable or quoted character string except the last is followed by an 015. For string variables only the actual data characters are sent. The first character sent for each string variable is the formpointed character. All characters through the logical length of the string variable will be sent.

Even vertical record parity (VRC) is generated on each character sent. The LRC parity generated is the exclusive or sum of every character sent after the STX up to and including the ETX.

3.9.2 RECEIVE (3)

The RECEIVE instruction receives data transmitted from another 2200 over a data line through an internal 202 modem at 1200 baud. The data received is entered into string variables appearing in the list following the RECEIVE instruction. The first item in the list may be a number between 0 and 255 decimal which indicates how many seconds the program should wait for an STX (start of message) to be received. If the number is 0 or if there is no number, the program will wait indefinitely.

As the data is entered into the variables, the formpointer of each string variable is set to one, and the characters are stored consecutively into the strings starting at the beginning of each string. Any quoted character strings that are sent must be received as string variables.

If the message received contains more items than the list, the remaining unused variables will be disregarded except in checking the LRC. If the list contains more variables than were in the message, the remaining variables will have their lengths and formpointers set to zero. If any variable from the message contains more characters than the physical length of the list variable, an EOS condition is set, but the rest of the message is still received.

Example:

```
RECEIVE 3,NBR,MSG
RECEIVE 0,A,B,C
RECEIVE MSG1,MSG2
```

As the characters are received, each character is checked for even vertical record parity (VRC). Also LRC parity is generated over every character received after the STX up to and including the ETX. The sum generated internally is compared to the LRC received at the end of the message. If any characters are received that do not have even VRC, or if the LRC received does not equal the LRC generated internally, the PARITY and ERROR conditions are set.

If the STX (start of message) is not received within the time limit set by the RECEIVE instruction, the TIME and ERROR conditions are set. If more than 20 milliseconds elapses between characters after the STX has been received the TIME and ERROR conditions will also be set. If any of the TIME, PARITY, or ERROR conditions are set, the entire list of variables in the RECEIVE list will have their lengths and formpointers set to zero.

The RECEIVE instruction may be aborted by holding down the KEYBOARD and DISPLAY keys simultaneously.

The SEND and RECEIVE instruction statements may be continued to the next line if a colon (:) is the terminating character of the instruction. The colon replaces the comma separating the last entry of the first line from the first entry on the second line. The first entry of the second line should begin in the instruction field.

Example:

```
XMIT SEND NBR,MSG1,MSG2,MSG3,MSG4:
     MSG5,MSG6
RECV RECEIVE 4,NAME,POSN,ADR,SSN,CODE:
     ITEM,COMP
```

3.9.3 WAIT (3)

The WAIT instruction causes the program to wait the number of seconds indicated by the number following the WAIT instruction. During the wait loop, the program continues to look for ringing present and the KEYBOARD and DISPLAY keys. If ringing is detected, the WAIT is stopped and the RING trap transfer is executed if it has been set. If the KEYBOARD and DISPLAY keys are simultaneously depressed, the program will abort.

Example:

```
WAIT 5
```

causes the program to wait 5 seconds.

3.9.4 DIAL (3)

The DIAL instruction causes the program to dial the number found in the string variable following the DIAL instruction. An asterisk in the string will cause a delay of 2 seconds. Other than an asterisk, all characters except the numbers 0 through 9 will be ignored.

Example:

```
NBR INIT "9*696-4520"
    DIAL NBR
```

3.9.5 CONNECT (3)

The CONNECT instruction causes the program to go offhook, and then waits for Data Coupler Ready status bit to come true.

Example:

```
CONNECT
```

3.9.6 DSCNCT (3)

the DSCNCT instruction causes the program to go onhook. The program then waits for five seconds before executing the next instruction.

Example:

```
DSCNCT
```

4.0 DATABUS SOURCE CODE EDITOR

The DATABUS mode of GEDIT should be used for preparation and editing of source data tapes. Some DATABUS Program Generation tapes have this program cataloged therein. If not, the GEDIT program and instruction manual should be obtained.

In addition to using GEDIT for DATABUS source code preparation, the text mode of GEDIT 1.4 and later versions contains an option which allows the user to generate DATABUS Write Edit records. These tapes may be read by the new tape format Databus Interpreters.

4.1 Databus Check List

The following check list may be used before compiling a program to prevent compile time errors.

Make sure:

1. Labels and variables have only six characters or less and are valid symbols.
2. There are not too many labels or variables in the program.
3. All labels and variables are defined, but not doubly defined. (Two labels or two variables must not have the same name).
4. All common variables are defined in exactly the same order and length as the variables in the other programs.
5. All instructions are spelled correctly.
6. There are no unmatched quote signs and no cursor positions off the screen.
7. The program does not exceed the allotted user space.

5.0 DATABUS COMPILER OPERATION

The Databus Compilers generate object programs which can be interpreted by the Databus Interpreters. The object program can also be cataloged by the operating system so that once a program has been compiled, it can be run any number of times without being recompiled.

The compiler makes one pass over the symbolic source code. All statements are checked for syntax and form. If any errors are found, flags are given. As the program is compiled, a program listing and an object program on tape are generated.

The compiler assigns numeric values to the various instructions and operands. Each instruction mnemonic has an octal value assigned to it as do the various conditions, events, units, variables and labels.

Two symbol tables are generated by the compiler, one for variables and one for labels. The object code values assigned to variables and the pointers determine in which table the entry can be found, 1 if variable and 0 if label. The low order seven bits determine the position of the symbol in the table. The last two bytes of each entry are output as part of the object code, forming lookup tables for the labels and variables mentioned above.

All variables are defined by directives, that is they must appear in the label field of directive instructions. Any symbols which appear in the label field of executable instructions are placed in the label table. All directives must appear before the first executable instruction in the program. Any directives which appear after the first executable instruction are given I-flags and their labels are placed in the label table instead of the variable table. Therefore, any references to these symbols will be flagged undefined.

In short, variables cannot be forward referenced, but labels can. Since the compiler makes only one pass over the source code, all labels are entered into the label table when found in the label or operand field of an instruction. No U-flags are given for undefined labels until the end of compilation when the symbol tables are output as part of the object code.

All undefined variables are entered into the variable table and flagged at the end so that the symbol tables output at the end of the listing will show all undefined symbols.

The following errors can occur during compilation:

1. D The D flag means DOUBLE DEFINITION. It is flagged if a label or variable has been defined to more than one value during compilation. In that case, it has the first value.

2. I The I flag means INSTRUCTION MNEMONIC UNKNOWN. The instruction was not an acceptable instruction code. In this case a 345 is inserted for the instruction.

3. E The E flag means that an error has occurred in the operand field of a statement or some unrecognizable character appeared in the wrong place. In this case a zero is substituted for

the operand or whatever was unrecognizable.

4. U The U flag means UNDEFINED SYMBOL. It is used whenever a symbol is referenced and is not defined.

OVERFLOW - This message is given if the user program exceeds its allotted space.

DICTIONARY FULL - This message is given if the user program has too many labels or variables.

Operating the Compiler:

Place a symbolic source tape generated by the Editor in the front deck.

Run the Databus Compiler. Several options will be made available to the user. The following questions will be asked.

PRINT? Type YES if a hard copy listing is desired; otherwise type NO.

DISPLAY? Type YES if a CRT display is desired; otherwise type NO.

CODE? Type YES if the object code is desired in the listing or display; otherwise type NO. (Code adds 18 columns to the listing.)

HEADING: Type in the heading. (This option is given only when a listing is desired.)

The source tape will be rewound and then compiled. At the end of compilation the object code block on the rear tape is copied to the front deck. The operating system is reloaded and comes up running.

6.0 RUNTIME OPERATION

Before running a DATABUS program, two programs must be cataloged onto a CTOS tape. The first is the interpreter. It is most convenient to have it as the first program on the tape because it is a lengthy file and passing over it should be avoided as much as possible. The second is a Databus program whose name must be MASTER, and it is most convenient to have it as the second program on the tape. The standard MASTER program will simply ask the operator for the name of the program he wishes to run, but any DATABUS program could be put in its place.

To start a run, CTOS must first be loaded (the catalog information is essential to the interpreter's operation) and then the interpreter must be run with the RUN command. This will cause the MASTER program to be loaded and executed. This action also occurs whenever execution of a program is terminated (a STOP statement executed or program fault). The Databus MASTER program will ask for the name of a program to be run. Typing a name not in the CTOS catalog will cause an error message to be displayed and for the name to be requested again. Typing the name of a non-DATABUS program will cause it to be loaded and executed if it does not overlay the first 28 bytes of the main execution loop of the interpreter (see the various listings for specific addresses of the label START) overlay of these locations will either cause execution to begin at START+7 or complete confusion. Typing the name of a DATABUS program will cause it to be loaded and executed unless the compiler generated some error messages, in which case an error abort will be made.

Once the program is running, execution may be terminated for a number of reasons. Execution of a STOP statement is equivalent to a CHAIN to the MASTER program. All other terminations will first print an error message of the format:

```
     (error message)  AT nnnnn
```

nnnnn will be the statement number (number that appears to the left of the statement on the compiler listing) on the statement <u>after</u> the one which is at fault. After this message is displayed, an EOF mark will be written on any deck which is in write mode and a CHAIN to the MASTER program will be performed. If an EOF is written on the front deck, the tape will be left positioned just before it.

A list of the error messages and their meanings follows:

CODE An attempt was made to run an object file which was generated from a source file that the compiler found at fault.

ABORT Both the KEYBOARD and DISPLAY keys were depressed. The statement before nnnnn was the last one executed.

BOP An undefined operation code was found at location nnnnn. This can happen only if there is a software error in the DATABUS compiler or interpreter system, if there is a hardware error, or if the interpreter has been destroyed by a non-DATABUS program.

MODE A tape I/O statement before statement nnnnn other than WRITE or WEOF was executed while the given deck was in write mode. An EOF will be written on that deck during the abort procedure.

BAD TAPE During the tape I/O operation before statement nnnnn a record of illegal format was read. This may be caused by parity errors or by trying to read a tape generated by some other program (e.g., the source tape from an EDIT operation was left in the front deck).

BUFUL During the tape write I/O operation before statement nnnnn, more than 240 bytes were written to cassette tape in old tape format records or 249 bytes in new tape format records, or 1057 bytes to magnetic tape. The tape write will not occur but an EOF mark will be written to the tape during the abort procedure.

EOF An end-of-file condition arose during the tape I/O operation before statement nnnnn and the trap was not set.

EOT An end-of-tape condition arose during the tape I/O operation before statement nnnnn and the trap was not set.

FORM During the tape read before statement nnnnn, either an item of the wrong type was read (string into number or visa versa) or more items appeared in the statement list than were on the tape record, and the trap was not set. The FORM trap is used in the old tape format interpreters.

RFAIL During the tape read before statement nnnnn, either an item of string type was read into a numeric variable or the tape record contained more than 249 data characters. The RFAIL trap is used in the new tape format interpreter.

CFAIL Execution of the CHAIN statement before statement nnnnn failed to find the requested name in the CTOS catalog.

7.0 FILES

The DATABUS facility includes two sequential files for mass storage. A single file on each cassette implements this, with the CTOS system scratch area (file 32) being used on the rear deck. Usually the rear deck will be used for temporary storage (as it is attached to the system tape) and the front deck will be used for changeable data files. The information on the front deck is contained in a CTOS type file zero (the same that is used by the editor in storing source information).

OLD TAPE FORMAT

In the old cassette tape format records, the variables on tape look like they do in memory. That is, the length, formpointer, and ETX of string variables and the 0200 and ETX of numeric variables are written in the tape records. The maximum length of the old format record is 244 characters - 4 CTOS header and 240 data characters. The format is CTOS numeric and appears as follows:

```
/(303)/(074)/XP/CP/data....../
```

The data consists of one or more variables, of string or numeric type.

A string variable in the old format may look like one of the following two:

```
a)  /LENGTH/FORMPOINTER/STRING(UP TO 127 CHARS)/ETX(203)/
```

The LENGTH is between 1 and 127 and equals the number of characters in the string on tape.

```
b)  /(000)/(000)/ETX(203)/
```

This is a null string.

For example, a string variable dimensioned to 15 with JOHN P BROWN entered into it looks as follows:

```
/(014)/(001)/J/O/H/N/ /P/ /B/R/O/W/N/ETX(203)/
```

A numeric variable in the old format will look like the following:

```
/(200)/number/ETX(203)/
```

For example, a numeric variable dimensioned to 8 characters as FORM 5.2 looks as follows:

```
/(200)/3/9/5/7/4/./9/8/ETX(203)/
```

A new tape format has been introduced to Databus. This format will be used in GEDIT, ASM, and the terminal emulators. The new format interpreters should be used in new applications. Since GEDIT can read Databus tapes now, the data tapes may be edited.

NEW TAPE FORMAT

The new cassette tape format looks like the old GEDIT format, except that every physical record is terminated by a 3. The Databus version of the new tape format is called the "Write-Edit" format. The maximum length of the new tape format record is 255 characters - 4 CTOS header, 249 data characters, an 015 (Logical End of Record - LEOR), and a 003 (Physical End of Record - PEOR). The record is CTOS numeric and appears as follows:

```
/(303)/(074)/XP/CP/data......./(015)/(003)/
```

The data consists of one or more variables, of string or numeric type. Since only the data characters of the variables are written to tape (length, formpointer, 0200, and ETX are deleted), each string variable will have a length equal to its defined physical length. WRITE begins writing at the first character of string variables and continues to the physical end of the string. Blanks are written for all characters after the logical end of the string.

A string variable dimensioned to 15 characters looks as follows on tape:

```
/J/O/H/N/ /P/ /B/R/O/W/N/ / / /
```

A numeric variable dimensioned to 8 characters as FORM 5.2 looks like the following on tape:

```
/3/9/5/7/4/./9/8/
```

When writing to tape in either old or new tape format, all list items are transferred into a buffer and the parity sums are generated. Then the block is written to tape. Writing larger blocks is advisable as increased tape efficiency results.

In Databus 3, 4, and 5 there is no facility for handling numeric variables internally. In these systems all numeric variables read from cassettes must be read into string variables and only string variables may be written. This is true for both old and new cassette tape formats.

INDUSTRY COMPATIBLE MAGNETIC TAPE FORMAT

In addition to the cassete mass storage, Databus 3 allows mass storage on two 7-Track or two 9-Track tape units. Each tape may have many sequential files. Each file is separated by an EOF mark and has no file number.

A tape unit with multiple files would appear as follows:

/File/EOF/File/EOF/File/EOF---/EOF/EOF/

EOF - End Of File.

Two EOF marks indicate the end of data on a tape.

A file looks as follows:

/Data/IRG/Data/IRG/Data/.../EOF/

IRG - Interrecord Gap.

Each data record is written by one WRITE statement, and read by one READ statement. In reading and writing to tape the hardware buffer in the tape controller is used so that 1057 characters is the maximum number of characters that may be written in one record.

Only character strings may be written and read on tape. The strings are written much like the new cassette tape format. The entire physical length of strings is written, except that writing begins at the formpointed character. Blanks are written for all characters after the logical end. Only the actual data characters are written to tape. There is no CTOS-provided header or parity, and there are no delimiters between strings. The tape controllers write and check a hardware generated parity, but this function is automatic and does not affect the program or format in any way.

8.0 CHAINING TO NON-DATABUS PROGRAMS

CTOS Databus uses the CTOS symbolic loader to perform the actual loading function. A CALL is made to MLOAD$. After a return is made from this call, Databus checks the user program starting location in RUN$ of the Loader. Each compiler always generates an object file with the same starting location. This location varies in the different compilers. The Interpreter then assumes that if the object file just loaded has this starting location, then the object file must be Databus object code. If the starting location was something different, Databus simply jumps to RUN$. Therefore, to CHAIN to a non-Databus program, make a CHAIN to that program, providing that its starting location is not the Databus user program starting location, and that it does not overlay the routine residing in the first 28 bytes (START thru START+28) of the Interpreter. This is the section which checks the user program starting location.

Example:

```
NONDAT INIT "NONDAT"
       CHAIN NONDAT
```

If the non-Databus program resides within the Databus user area, then it can chain back to a Databus program by simply loading DE with the address of the program name string (using MLOAD$ rules) and jump back to START+4 for Databus 1, 2, and 3 and START+2 for Databus 4 and 5.

Example:

```
RETDAT DC "RETDAT"
       DE RETDAT     No Interpreter Overlay
       JMP START+4   Load Databus 2 Program
START  EQU 03500
```

If the non-Databus program does not reside in the Databus user area, then it must reload the Databus Interpreter and jump to START (which will cause the MASTER program to be executed) or load DE with the address of the program name string and jump to START+4 for Databus 1, 2, and 3 or START+2 for Databus 4 and 5.

Example:

```
   DB2INT DC   "DB2INT"  Interpreter Overlayed
          DE   DB2INT
          CALL MLOAD$    Load Interpreter
   START  JMP  START     Load and Execute MASTER
          EQU  03500
   RETDAT DC   "RETDAT"  Interpreter Overlayed
   DB2INT DC   "DB2INT"
          DE   DB2INT
          CALL MLOAD$    Load Interpreter
          JMP  START+4   Load Databus 2 Program
   START  EQU  03504
```

	DB1	DB2	DB3	DB4	DB5
Databus User Program Starting Location	017044	017044	02266	05744	06654
START-Interpreter Starting Location	05000	03500	02350	01000	01400

See the individual Databus Sections for the user areas of each.

9.0 INTERPRETER INTERNAL OPERATION

The interpreter fetches and executes instructions (statements) much like a computer. It contains within its working storage area the equivalent of the program counter, condition register, instruction register, and other miscellaneous items. The basic instruction format is one byte broken into two fields:

N	N	0	0	0	0	0	0

The NN bits indicate the number of bytes in the instruction. For I/O operations, this number is either one or two and the rest of the instruction is read by scanning for the list terminator. This number is never zero.

The 000000 bits indicate which operation is to be performed. This number provides an index into an address table which causes the interpreter to execute the proper subroutine to perform that instruction.

Operands and labels are addressed by single bytes. Labels have their sign bit clear and operands have them set. The remaining seven bit numbers index into address tables (one for labels and a different one for operands) which are generated by the compiler at the end of compilation. Because of this, the compiler only needs to be a one pass process. Since these tables are placed after the user's code, they may be located anywhere, so the compiler cranks out two other addresses in the interpreter working storage area which point to the beginning of each table. Thus, a typical instruction execution sequence would be as follows:

a) Get the byte pointed to by the PC and increment the PC.

b) Get the operand pointed to by the PC and increment the PC.

c) Branch to the correct routine based on the value of the right six bits of the opcode. The correct address is obtained by multiplying the right six bits by two and adding the result to the execution routine address table. Load the address of the routine from the table and jump to it.

d) The instruction would take the operand number, isolate the right seven bits, multiply it by 2, add it to the base address of the operand table, load the address of the variable or label from the table and perform some operation upon the variable or label pointed to.

In DISPLAY, KEYIN, PRINT, and SEND, immediate characters (quoted items) are denoted by not having their sign bit set. These characters are simply printed unless they have a special control function for the instruction in which they appear. The controls fall in the group between 0 and 37 octal.

10.0 CONFIGURATION

The Databus Interpreters and Compilers may be configured to run with a local or remote printer at any speed in different size and version machines. The configuration programs are named DBnCC and DBnIC, where n is the number of the Databus compiler or interpreter to be configured. DBnCC is the compiler configurator, and DBnIC is the interpreter configurator.

Execution of these programs causes a request for a series of responses from the user. After the questions have been answered, the first block of the corresponding interpreter or compiler will be overlayed. The compiler cataloged must be named DBnCMP and the interpreter must be named DBnINT in the CTOS catalog for the overlay to occur. The "n" in these names must match the "n" in the configurator names.

Any or all of the following questions may be asked by the various configurators.

QUESTION	RESPONSE
For DBnIC:	
1) Object Machine Version (1,2)?	Answer 1 or 2 if Interpreter and user program is to be run in a Version 1 or 2 respectively.
2) Local or Remote Printer?	Answer L for Local Printer, i.e., 2200/P or 2200/LP. Answer R for Remote Printer, i.e., 3300/P, teletype, or any printer which requires a communications interface. Answer either if no printer is available.
3) Remote Printer Speed?	Asked only if R was the response to the previous question. Type in the required baud speed of the printer used. Type 300 for 3300/P, 110 for 100 w.p.m. Teletype.
4) Local or Servo Printer?	Answer L for Local Printer, or S for Servo Printer. Answer either if no printer is available.

For DBnCC:

5) Object Machine Size (8,12,16)?

Type in the size of the machine in which the Interpreter and user program will be run. This will define the user area.

6) Compiler Machine Size (8,12,16)?

Type in the size of the machine in which the user program will be compiled. This will define the number of labels and variables allowed.

7) Local or Remote Printer?

Answer L for Local Printer, i.e., 2200/P or 2200/LP. Answer R for Remote Printer, i.e., 3300/P, teletype, or any printer which requires a comm interface. Answer either if no printer is available.

8) Remote Printer Speed?

Asked only if R was the response to the previous question. Type in the required baud speed of the printer used. Type 300 for 3300/P, 110 for teletype. Baud speed equals 10 times the number of characters/second.

9) Local or Servo Printer?

Answer L for Local Printer, or S for Servo Printer. Answer either if no printer is available.

The following shows which of the above questions the different Databus Configurators ask:

Databus 1 -		(2)	(3)		(5)	(6)	(7)	(8)	
Databus 2									
Version 3 -		(2)	(3)		(5)	(6)	(7)	(8)	
Version 4 -				(4)	(5)	(6)			(9)
Databus 3 -	(1)	(2)			(5)	(6)	(7)	(8)	
Databus 4 -		(2)	(3)			(6)	(7)	(8)	
Databus 5 -		(2)	(3)				(7)	(8)	

When the configurator is completed, it will display DONE on the screen. The time before DONE is displayed may be considerable. After the message is displayed, CTOS will be reloaded.

11.0 CTOS DATABUS SUMMARIES

The following lists of definitions and input/output controls are referred to in each Databus instruction summary.

DATABUS DEFINITIONS:

address — Refers to the location in memory of assembly language subprogram to be called. May be octal or decimal.

character string — Any string of alphanumeric characters.

condition — The result of operations used in conditional transfer of control operations.

LESS,EQUAL,ZERO,OVER - result of any arithmetic operation
LESS,EQUAL,ZERO,EOS - result of any string operation
PARITY,TIME,ERROR,EOS- result of any RECEIVE operation

event — The occurrence of end of file, end of tape, data type error, tape read error, program chain failure, or ring detect.

EOF(unit)
EOT(unit)
FORM(unit)
RFAIL(unit)
CFAIL
RING

index — Refers to one of the eight possible one byte indexes, used for all arithmetic operations, I0 through I7.

label — A name assigned to a statement.

list — A list of variables, quoted character strings, or controls appearing in an input/output type of instruction.

literal — A quoted alphanumeric character or a number. The number may be octal or decimal as long as it is between 0 and 127 decimal.

n — Refers to an integer between 0 and 127 decimal.

n.m — Refers to any number octal or decimal. Octal if it is preceded by a zero, up to 22 total digits including the decimal point.

nvar — A label assigned to a directive defining a numeric string variable.

size — A number defining the memory size of the Datapoint 2200 in which the user program and interpreter will be run. It may be 4, 6, 8, 12, or 16.

sval — A label assigned to a directive defining a character string variable, or a quoted alphanumeric character, or a number. This number may be octal or decimal as long as it is between 0 and 127 decimal.

svar — A label assigned to a directive defining a character string variable.

unit — A number defining a tape deck.

1 = Deck 1 (rear)
2 = Deck 2 (front)
3 = Mag Tape Deck (addr=226 octal)
4 = Mag Tape Deck (addr=113 octal)

456.23 — Refers to any octal or decimal number, up to 22 total digits.

DATABUS INPUT/OUTPUT CONTROLS:

CONTROL	APPLICABLE INSTRUCTION	FUNCTION
*Hn	KEYIN DISPLAY	Causes cursor to be positioned horizontally to the column indicated by the literal or numeric variable n, $1 \le n \le 80$.
*Vn	KEYIN DISPLAY	Causes the cursor to be positioned vertically to the row indicated by literal or numeric variable n, $1 \le n \le 12$.
*EL	KEYIN DISPLAY	Causes the c.r.t. screen to be erased from the current cursor position to the end of the line.
*EF	KEYIN DISPLAY	Causes the c.r.t. screen to be erased from the cursor position to the end of the screen.
*R	KEYIN DISPLAY	Causes the c.r.t. screen to roll up one line, losing the top line and setting the bottom line to blanks. (The cursor position does not move.)
*n	PRINT	Causes horizontal tab to the column indicated by the number n. (No action occurs on the local printer if the carriage is past the column indicated by n.)
;	KEYIN DISPLAY PRINT	Suppresses a new line function when occurring at the end of a list, i.e., the cursor or print carriage remains in the position indicated by the completion of the last list element.
"	KEYIN DISPLAY PRINT SEND	Any characters appearing between quotes are displayed, printed, or sent when encountered.
*F	PRINT	Causes the printer to be positioned to the top of form.
*L	PRINT	Causes a linefeed to be printed.
*C	PRINT	Causes a carriage return to be printed.

PROGRAM LENGTH

a) Numeric String Variables use two words plus one word for each string character (including decimal point and sign if negative).

b) Character String Variables use three words plus one word for each string character.

c) String Instructions except LOAD and STORE use two or three words depending on whether one or two variable names are required for the instruction.

d) Arithmetic Instructions except LOAD and STORE use three words. LOAD and STORE fall into the Control category for space allocation.

e) Control and Input/Output Instructions require one word for the command plus one word for each label, condition, event, variable, or unit used. Strings found in I/O instructions add one word per character. I/O controls which begin with an asterisk add one or two words for each occurrence (*C, *L, *F, *EL, *EF, *R use one word, all others use two). Every instruction which contains a list uses one additional word for the list terminator.

f) Two additional words are used for each label or variable.

11.1 Databus 1

11.1.1 Instruction Summary

```
Directives
  FORM n.m
  FORM "456.23"
  DIM n
  INIT "character strinq"
  FORM *n.m
  FORM *"456.23"
  DIM *n
  INIT *"CHARACTER STRING"

Control
  TRAP (label) IF (event)
  GOTO (label)
  GOTO (label) IF (condition)
  GOTO (label) IF NOT (condition)
  CALL (label)
  CALL (label) IF (condition)
  CALL (label) IF NOT (condition)
  RETURN
  RETURN IF (condition)
  RETURN IF NOT (condition)
  STOP
  STOP IF (condition)
  STOP IF NOT (condition)
  CHAIN (svar)
  BRANCH (nvar) OF (label list)

Numeric Variable Arithmetic
  ADD (nvar) TO (nvar)
  SUB (nvar) FROM (nvar)
  MULT (nvar) BY (nvar)
  DIV (nvar) INTO (nvar)
  MOVE (nvar) TO (nvar)
  COMPARE (nvar) TO (nvar)
  LOAD (nvar) FROM (nvar) OF (nvar list)
  STORE (nvar) INTO (nvar) OF (nvar list)

KEYBOARD, C.R.T., PRINTER I/O
  KEYIN (list)
  DISPLAY (list)
  PRINT (list)
  BEEP
  CLICK
  DSENSE
  KSENSE

Cassette Tape I/O
  READ (unit),(list)
  WRITE (unit),(list)
```

REWIND (unit)
BKSP (unit)
PREPARE (unit)
WEOF (unit)

11.1.2 Conditions

OVER
LESS
EQUAL
ZERO
EOS

11.1.3 Events

EOF1	
EOF2	
EOT1	
EOT2	
FORM1	Old Tape Format
FORM2	" " "
RFAIL1	New Tape Format
RFAIL2	" " "

11.1.4 User Area

<u>Interpreter Machine</u>

8K - 3400_8 bytes (1000_8 - 4377_8)
12K - 10000_8 bytes (20000_8 - 27777_8)
16K - 20000_8 bytes (20000_8 - 37777_8)

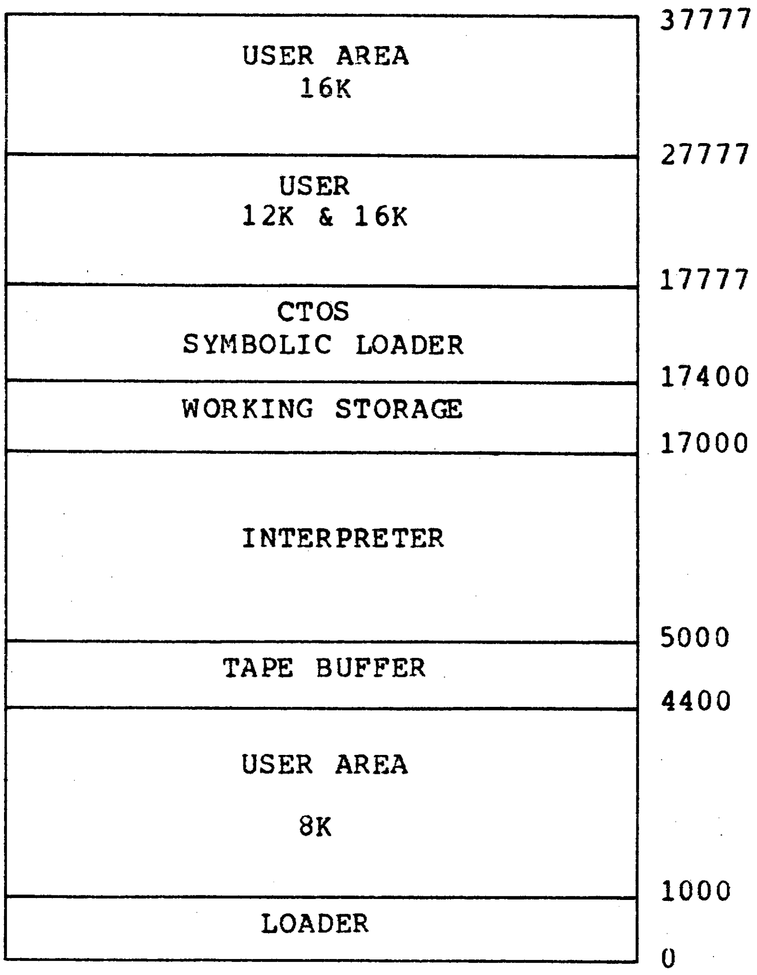
11.1.5 Dictionaries

<u>Compiler Machine</u>

8K - 100 labels, 100 variables
12K or 16K - 125 labels, 125 variables

11.1.6 Interpreter Internal Structure

Databus 1 is layed out in memory as follows:



11.1.7 Sample Programs

DATABUS 1 FILE INPUT PROGRAM

```
        . File Input Program
        .
        . Sample Databus 1 Program
        . This is a File Input and Updat Program.
        . It preps the front tape if it is a new file or
        . positions the front tape to after the last file
        . record input if it is an old file.
        . Allows user to type in file records, and then
        . writes the records to tape.  Four items plus a
        . command may be input.  The command is interpreted
        . as follows:
        .      0 -- The items are correct, so write them to
        .           tape
        .      1 -- All records have been input, so end the
        .           program
        .    <0 or >1 -- The items are incorrect, reinput
        .           them
        . When all information has been input, an end-of-
        . file is written to tape and execution returns to
        . the MASTER program.

01000   ZERO    FORM "0."
01004   ONE     FORM "1."
01010   CMND    FORM 1.
01014   LNBR    FORM 4.
01023   ITEM1   DIM 50
01110   ITEM2   DIM 50
01175   ITEM3   DIM 50
01262   ITEM4   DIM 50

01347   FILIN   REWIND 2
01351           DISPLAY *V1,*H1,*EF,"FILE INPUT PROGRAM"
                DISPLAY *V4,*H1,"LABEL NUMBER:"
01423           DISPLAY "ITEM1:"
01433           DISPLAY "ITEM2:"
01443           DISPLAY "ITEM3:"
01453           DISPLAY "ITEM4:",*V9,*H45,"CHECK:"
01476           KEYIN *V3,"NEW TAPE (0=NO,1=YES):",CMND
01532           COMPARE ONE,CMND
01535           GOTO PREP IF EQUAL
01540           TRAP INPUT IF EOF2
01543   SKIP    READ 2,LNBR
01547           GOTO SKIP
01551   INPUT   ADD ONE,LNBR
01554   REPEAT  DISPLAY *V4,*H15,LNBR,*V5,*H8,*EL:
01567           *V6,*EL,*V7,*EL,*V8,*EL
01601           KEYIN *V5,*H8,ITEM1,*V6,*H8,ITEM2,*V7,*H8,ITEM3
```

DATABUS 1 FILE INPUT PROGRAM

```
01622           KEYIN *H8,ITEM4,*V9,*H52,CMND
01634           COMPARE ONE,CMND
01637           GOTO TERM IF EQUAL
01642           COMPARE ZERO,CMND
01645           GOTO REPEAT IF NOT EQUAL
01650           WRITE 2,LNBR,ITEM1,ITEM2,ITEM3,ITEM4
01660           GOTO INPUT
01662   TERM    WRITE 2,LNBR,ITEM1,ITEM2,ITEM3,ITEM4
01672           WEOF 2
01674           REWIND 2
01676           STOP
01677   PREP    MOVE ZERO,LNBR
01702           PREPARE 2
01704           GOTO INPUT
01706           STOP

01707   FILIN
01711   PREP
01713   INPUT
01717   REPEAT
01721   TERM

01723   ZERO
01725   ONE
01727   CMND
01731   LNBR
01733   ITEM1
01735   ITEM2
01737   ITEM3
01741   ITEM4
```

DATABUS 1 FILE DISPLAY PROGRAM

- PROGRAM FILE DISPLAY

- DISPLAYS FILE FROM FILE INPUT PROGRAM

```
01000    LNBR    FORM 4
01006    ITEM1   DIM 50
01073    ITEM2   DIM 50
01160    ITEM3   DIM 50
01245    ITEM4   DIM 50

01332    START   REWIND 2
01334            DISPLAY *H1,*V1,*EF,"FILE DISPLAY PROGRAM":
01366            *H1,*V4,"LABEL NUMBER:"
01410            DISPLAY *H1,*V5,"ITEM1:",*H1,*V6,"ITEM2:":
01435            *H1,*V7,"ITEM3:"
01450            DISPLAY *H1,*V8,"ITEM4:"
01464            TRAP END IF EOF2
01467    LOOP    READ 2,LNBR,ITEM1,ITEM2,ITEM3,ITEM4
01477            DISPLAY *H15,*V4,*EL,LNBR,*H8,*V5,*EL,ITEM1:
01514            *H8,*V6,*EL,ITEM2
01523            DISPLAY *H8,*V7,*EL,ITEM3,*H8,*V8,*EL,ITEM4
01541    DWAIT   DSENSE
01542            GOTO DWAIT IF NOT EQUAL
01545            GOTO LOOP
01547    END     REWIND 2
01551            STOP

01552    START
01554    END
01556    LOOP
01560    DWAIT

01562    LNBR
01564    ITEM1
01566    ITEM2
01570    ITEM3
01572    ITEM4
```

11.2 DATABUS 2

11.2.1 Instruction Summary

Directives

```
FORM n.m
FORM "456.23"
DIM n
INIT "character string"
FORM *n.m
FORM "456.23"
DIM *n
INIT *"CHARACTER STRING"
```

Control

```
TRAP (label) IF (event)
GOTO (label)
GOTO (label) IF (condition)
GOTO (label) IF NOT (condition)
CALL (label)
CALL (label) IF (condition)
CALL (label) IF NOT (condition)
RETURN
RETURN IF (condition)
RETURN IF NOT (condition)
STOP
STOP IF (condition)
STOP IF NOT (condition)
CHAIN (svar)
BRANCH (nvar) OF (label list)
```

String

```
CMATCH (sval) TO (sval)
CMOVE (sval) TO (svar)
MATCH (svar) TO (svar)
MOVE (svar) TO (svar)
MOVE (svar) TO (nvar)
MOVE (nvar) TO (svar)
APPEND (svar) TO (svar)
RESET (svar) TO (sval)
RESET (svar) to (nvar)
RESET (svar)
BUMP (svar) by (literal)
BUMP (svar)
ENDSET (svar)
LENSET (svar)
TYPE (svar)
EXTEND (svar)
CLEAR (svar)
LOAD (svar) FROM (nvar) OF (svar list)
STORE (svar) INTO (nvar) OF (svar list)
```

```
Numeric Variable Arithmetic
  ADD (nvar) TO (nvar)
  SUB (nvar) FROM (nvar)
  MULT (nvar) BY (nvar)
  DIV (nvar) INTO (nvar)
  MOVE (nvar) TO (nvar)
  COMPARE (nvar) TO (nvar)
  LOAD (nvar) FROM (nvar) OF (nvar list)
  STORE (nvar) INTO (nvar) OF (nvar list)

Keyboard, C.R.T., Printer I/O
  KEYIN (list)
  DISPLAY (list)
  PRINT (list)
  BEEP
  CLICK
  DSENSE
  KSENSE

Cassette Tape I/O
  READ (unit),(list)
  WRITE (unit),(list)
  REWIND (unit)
  BKSP (unit)
  PREPARE (unit)
  WEOF (unit)
```

11.2.2 Conditions

```
OVER
LESS
EQUAL
ZERO
EOS
```

11.2.3 Events

```
EOF1
EOF2
EOT1
EOT2
FORM1     Old Tape Format
FORM2      "    "     "
RFAIL1    New Tape Format
RFAIL2     "    "     "
CFAIL
```

11.2.4 User Area

<u>Interpreter Machine</u>

8K - 2500_8 bytes (1000_8-3477_8)

12K - 10000_8 bytes (20000_8-27777_8)

16K - 20000_8 bytes (20000_8-37777_8)

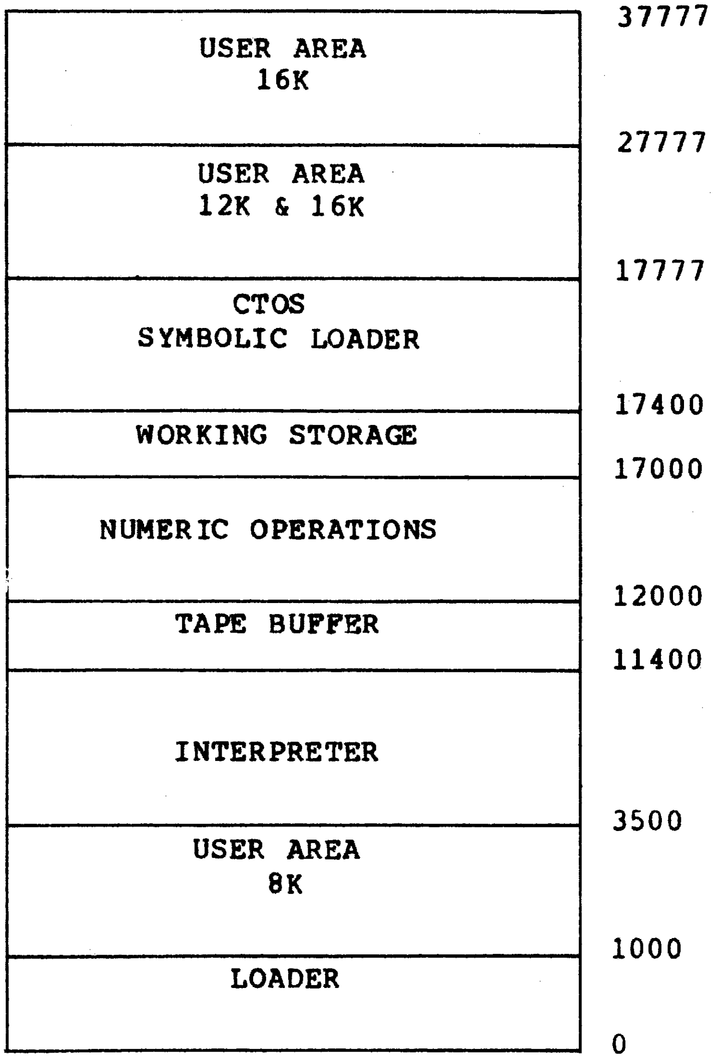
11.2.5 Dictionaries

Compiler Machine

8K - 100 labels, 100 variables
12K or 16K - 125 labels, 125 variables

11.2.6 Interpreter Internal Structure

Databus 2 is layed out in memory as follows:



11.2.7 Sample Programs

The sample Databus 2 programs included make up a simple file handling system. It is by no means complete, but serves to give an idea of what can be done with Databus 2.

A brief summary of each program will be given to aid in tracing through the programs. These programs include an update file entry program, a two tape merge of the update file into the master file, as well as programs to display and copy the two files.

UPDATE PROGRAM

1. Positions rear deck to Update File.
2. Allows user to type in the 5 fields of information for the update records.
3. As each field is input, it is appended to a buffer. Slashes are used as field delimiters.
4. Writes out the packed record to the update file.
5. If more update files need to be input goes to 2.
6. Otherwise writes a dummy record to indicate the end of the update file, and a physical end of file.
7. Chains back to the MASTER program.

UPDATE FILE DISPLAY PROGRAM

1. Reads update file records from the rear deck.
2. Displays each record exactly as it was written to tape, rolling up the screen as each entry is displayed.
3. When all records have been displayed, execution returns to the MASTER program.

TWO TAPE MERGE PROGRAM

1. Asks if front tape is a new master tape.
2. If the master is new, the front deck is prepped. The rest of the program treats the front deck the same whether it is an old or new master.
3. The rear deck is positioned to the update file.
4. Five records are read from the update file. The records were written with the name field first, and the merge is done in alphabetical name order.
5. The rear deck is then positioned to the end of the update file.
6. The smallest of the five update records is found.
7. The smallest record is then merged into the master tape. This is done in the following manner:

   a. The master tape records are read in one at a time.
   b. The master record is compared to the up-

date record. If the master record is smaller it is written to the update tape (now positioned after the end of the update file), and a new master record is read in and compared.

   c. If the update record is smaller, it is written to the update tape.
   d. Execution then returns to 6 where the next smallest update record is found until all 5 update records have been merged.

8. Once all 5 are merged, the rest of the master tape is copied to the rear deck.
9. The rear new master is copied back to the front tape.
10. The update file is then positioned after the last five update records are read in, and then five more records are read (or as many as are left).
11. Execution then returns to 5.
12. Once all update records are merged and the final master copied back to the front tape, the tapes are rewound, and execution returns to the MASTER program.

MASTER FILE DISPLAY PROGRAM

1. Rewinds the front master tape.
2. Reads in a record.
3. Unpacks the record into five fields. The unpacking is done by a character match, searching for a slash, the field delimiter.
4. When all fields are unpacked, the information is displayed on the screen.
5. Execution then returns to 2 until all records have been read and displayed.
6. Execution returns to the MASTER program.

COPY PROGRAM

1. Copies records (maximum of 127 chars) from front deck to rear, and rear deck to front. The records are written to file 32 on the rear deck, and file 0 on the front deck.
2. When all records have been copied, execution returns to the MASTER program.

MASTER FILE DISPLAY PROGRAM

```
          . PROGRAM LIST
          .
          . DISPLAYS MASTER FILE
          . READS IN RECORD FROM TAPE, UNPACKS THE
          . DATA INTO FIVE FIELDS, AND DISPLAYS THE
          . FIELDS ON THE SCREEN.

01000     NAME         DIM 20
01027     ADDR         DIM 30
01070     SSN          DIM 11
01106     BUSNES       DIM 20
01135     CCODE        DIM 4
01145     SLASH        INIT "/"
01150     BUFF         DIM 89
01304     TEMP         DIM 30
01345     COUNT        FORM "01"
01351     ONE          FORM "1"
01354     SIX          FORM "6"
01357     EXIT         INIT "MASTER"
01370     START        DISPLAY *H1,*V1,*EF,*H15,"MASTER FILE DISPLAY"
01424                  DISPLAY *H1,*V2,"FRONT TAPE MASTER?";
01454                  KEYIN TEMP
01457                  REWIND 2
01461                  TRAP END IF EOF 2
01464     RD           READ 2,BUFF
01470                  MOVE ONE,COUNT
01473                  CLEAR TEMP
01475     LOOP         CMATCH BUFF,SLASH
01500                  GOTO NEXT IF EQUAL
01503                  EXTEND TEMP
01505                  GOTO NEXT IF EOS
01510                  CMOVE BUFF,TEMP
01513                  BUMP BUFF
01515                  GOTO NEXT IF EOS
01520                  GOTO LOOP
01522     NEXT         RESET TEMP
01525                  STORE TEMP INTO COUNT OF NAME,ADDR,SSN:
01533                  BUSNES,CODE
01536                  ADD ONE,COUNT
01541                  COMPARE COUNT,SIX
01544                  GOTO DISPLY IF EQUAL
01547                  CLEAR TEMP
01551                  BUMP BUFF
01553                  GOTO LOOP IF NOT EOS
          .
          . DISPLAY ALL FIVE FIELDS ON SCREEN
          .
01556     DISPLY       DISPLAY *H1,*V5,*EF,"NAME:",NAME
01574                  DISPLAY *H1,*V6,"ADDRESS:",ADDR
01614                  DISPLAY *H1,*V7,"SOCIAL SECURITY #:",SSN
01646                  DISPLAY *H1,*V8,"COMPANY:",BUSNES
01666                  DISPLAY *H1,*V9,"CUSTOMER CODE:",CCODE
01714                  CLEAR NAME
```

MASTER FILE DISPLAY PROGRAM

```
01716                  CLEAR ADDR
01720                  CLEAR SSN
01722                  CLEAR BUSNES
01724                  CLEAR CCODE
01726                  GOTO RD
01730   END            CHAIN EXIT
01732                  STOP

01733   START
01735   END
01737   RD
01741   LOOP
01743   NEXT
01745   DISPLY

01747   NAME
01751   ADDR
01753   SSN
01755   BUSNES
01757   CCODE
01761   SLASH
01763   BUFF
01765   TEMP
01767   COUNT
01771   ONE
01773   SIX
01775   EXIT
```

DATABUS TWO TAPE MERGE PROGRAM

```
         . PROGRAM DATABUS SORT PROGRAM
         .
         . MERGE PROGRAM
         .
         . READS IN UPDATE TAPE ON REAR DECK 5 RECORDS AT
         . A TIME AND MERGES THEM INTO MASTER ON FRONT DECK.
         . IF THE MASTER TAPE IS NEW, THE UPDATE TAPE IS
         . SORTED AND WRITTEN TO THE MASTER.
         .
01000    N1     DIM 89
01134    N2     DIM 89
01270    N3     DIM 89
01424    N4     DIM 89
01560    N5     DIM 89
01714    N5     DIM 89
02050    MASTER DIM 89
02204    TST    DIM 89
01240    NL     INIT "^^^^^^^^^^^^^^^^^^^^^"
02367    DUMMY  INIT "**WEOF**"
02402    EXIT   INIT "MASTER"
02413    FLAG   FORM "0"
02416    COUNT  FORM "01"
02422    SMALL  FORM "0"
02425    CNTSV  FORM "00"
02431    ONE    FORM "1"
02434    ZERO   FORM "0"
02437    TEMP   DIM 2
02444    RECORD FORM "0000"
02452    CNT    FORM "00"
02456    SIX    FORM "6"
02461    TEN    FORM "10"
         .
02465    START  DISPLAY *H1,*V1,*EF,*H15,"TWO TAPE MERGE PROGRAM"
02524           DISPLAY *H1,*V3,"READS IN UPDATE TAPE ";
02557           DISPLAY "AND MERGES IT INTO MASTER TAPE ";
02617           DISPLAY "IN ALPHABETICAL ORDER"
02646    ASK    DISPLAY *H1,*V4,*EL,"NEW MASTER?";
02670           KEYIN TEMP
02673           REWIND 1
02675           CMATCH TEMP,"Y"
02700           GOTO REWD IF NOT EQUAL
02703           PREPARE 2
02705           WEOF 2
02707    REWD   REWIND 2
         .
02711    SORT   MOVE NL,N5
02714    RD     READ 1,TST
02720           MATCH DUMMY,TST
02723           GOTO SETFLG IF EQUAL
02726           STORE TST INTO COUNT OF N1,N2,N3,N4,N5
02737           ADD ONE,COUNT
02742           COMPARE SIX,COUNT
```

```
02745          GOTO RD IF LESS
        .
02750   CNT    COMPARE COUNT,ONE
02753          GOTO END IF EQUAL
02756          MOVE COUNT,CNTSV
02761          SUB ONE,CNTSV
        .
02764   FEOF   READ 1,MASTER
02770          MATCH MASTER,DUMMY
02773          GOTO FEOF IF NOT EQUAL
02776          CLEAR MASTER
        .
03000   M1     MOVE ONE,COUNT
03003   FIND   LOAD TST FROM COUNT
03014          MATCH N5,TST
03017          CALL MOVE
03021          ADD ONE,COUNT
03024          COMPARE CNTSV,COUNT
03027          GOTO FIND IF LESS
03032          GOTO FIND IF EQUAL
        .
03035          TRAP MOVSTR IF EOF2
03040          RESET MASTER
03043          GOTO MRG IF NOT EOS
03046   MERGE  READ 2,MASTER
03052   MRG    MATCH N5,MASTER
03055          GOTO MOVSTR IF NOT LESS
03060          WRITE 1,MASTER
03064          GOTO MERGE
        .
03066   MOVSTR WRITE 1,N5
03072          MOVE NL,N5
03075          STORE NL INTO SMALL
03106          ADD ONE,CNT
03111          COMPARE CNTSV,CNT
03117          MOVE ZERO,CNT
03122          TRAP COPY IF EOF2
03125          RESET MASTER
03130          GOTO TRNSFR IF EOS
03133          WRITE 1,MASTER
03137   TRNSFR READ 2,MASTER
03143          WRITE 1,MASTER
03147          GOTO TRNSFR
        .
03151   COPY   WEOF 1
03153          REWIND 1
03155          PREPARE 2
        .
03157   SRCH   READ 1,MASTER
03163          MATCH MASTER,DUMMY
03166          GOTO SRCH IF NOT EQUAL
        .
03171          TRAP SETUP IF EOF1
```

DATABUS TWO TAPE MERGE PROGRAM

```
03174   RDWR    READ 1,MASTER
03200           WRITE 2,MASTER
03204           GOTO RDWR
        .
03206   SETUP   WEOF 2
03201           REWIND 2
03212           REWIND 1
03214           COMPARE FLAG,ONE
03217           GOTO END IF EQUAL
03222           ADD CNTSV,RECORD
03225           MOVE ZERO,COUNT
03230   RECRD   READ 1,MASTER
03234           ADD ONE,COUNT
03237           COMPARE COUNT,RECORD
03242           GOTO RECRD IF NOT EQUAL
03245           MOVE ONE,COUNT
03250           CLEAR MASTER
03252           GOTO SORT
        .
03254   END     CHAIN EXIT
        .
03256   MOVE    RETURN IF NOT LESS
03260           RETURN IF EQUAL
03262           LOAD N5 FROM COUNT OF N1,N2,N3,N4,N5
03273           MOVE COUNT TO SMALL
03276           RETURN
        .
03277   SETFLG  MOVE ONE,FLAG
03302           BKSP 1
03304           GOTO CNT
03306           STOP

03307   START
03311   ASK
03313   REWD
03315   SORT
03317   RD
03321   SETFLG
03323   CNT
03325   END
03327   FEOF
03331   M1
03333   FIND
03335   MOVE
03337   MOVSTR
03341   MRG
03343   MERGE
03345   COPY
03347   TRNSFR
03351   SRCH
03353   SETUP
03355   RDWR
03357   RECRD
```

DATABUS TWO TAPE MERGE PROGRAM

03361	N1
03363	N2
03365	N3
03367	N4
03371	N5
03373	N5
03375	MASTER
03377	TST
03401	NL
03403	DUMMY
03405	EXIT
03407	FLAG
03411	COUNT
03413	SMALL
03415	CNTSV
03417	ONE
03421	ZERO
03423	TEMP
03425	RECORD
03427	CNT
03431	SIX
03433	TEN

UPDATE FILE DISPLAY PROGRAM

```
         . PROGRAM LIST
         .
         . LIST UPDATE FILE
         . READS IN RECORDS FROM SCRATCH FILE ON REAR DECK
         . AND DISPLAYS THEM ON THE SCREEN
         .
01000    BUFF    DIM 89
01134    EXIT    INIT "MASTER"
         .
01145    START   REWIND 1
01147            TRAP END IF EOF1
01152    RD      READ 1,BUFF
01156            DISPLAY *H1,*V12,*EL,BUFF
01166            GOTO RD
         .
01170    END     CHAIN EXIT
01172            STOP

01173    START
01175    END
01177    RD

01201    BUFF
01203    EXIT
```

DATABUS UPDATE PROGRAM

```
        . UPDATE PROGRAM
        .
        . ALLOWS USER TO TYPE IN DESIRED INFORMATION.
        . THE DATA IS THEN PACKED AND WRITTEN OUT TO TAPE.
        . THE SCRATCH FILE ON THE REAR DECK IS USED FOR
        . THE UPDATE FILE.
        .
        .
01000   NAME    DIM 20
01027   ADDR    DIM 30
01070   SSN     DIM 11
01106   BUSNES  DIM 20
01135   CCODE   DIM 4
01144   TEMP    DIM 10
01161   UPDTE   INIT "UPDATE"
01172   END     INIT "END"
01200   EXIT    INIT "MASTER"
01211   SLASH   INIT "/"
01215   BUFF    DIM 89
01351   DUMMY   INIT "**WEOF**"
01364   START   DISPLAY *H1,*V1,*EF,*H15,"UPDATE PROGRAM"
01413           DISPLAY *H1,*V2,"TYPE IN THE REQUESTED INFO."
01464           PREPARE 1
        .
        . KEYIN INFORMATION FOR UPDATE RECORDS
        .
01466   UPDAT   KEYIN *H1,*V5,*EF,"NAME (LAST,FIRST):",NAME
01521           APPEND NAME,BUFF
01524           APPEND SLASH,BUFF
01527           KEYIN *H1,*V6,"ADDRESS:",ADDR
01547           APPEND ADDR,BUFF
01552           APPEND SLASH,BUFF
01555           KEYIN *H1,*V7,"SOCIAL SECURITY NUMBER:",SSN
01614           APPEND SSN,BUFF
01617           APPEND SLASH,BUFF
01622           KEYIN *H1,*V8,"COMPANY:",BUSNES
01642           APPEND BUSNES,BUFF
01645           APPEND SLASH,BUFF
01650           KEYIN *H1,*V9,"CUSTOMER CODE:",CCODE
01676           APPEND CCODE,BUFF
01701           RESET BUFF
        .
        . WRITE BUFFER TO UPDATE FILE
        .
01704           WRITE 1,BUFF
01710           CLEAR BUFF
        .
        . SEE IF END OF UPDATE OR MORE INFO
        .
01712   ASK     KEYIN *H1,*V11,"UPDATE OR END?",TEMP
01740           MATCH TEMP,UPDATE
01743           GOTO UPDAT IF EQUAL
01746           MATCH TEMP,END
```

```
01751              GOTO ASK IF NOT EQUAL
          .
          . IF END THEN WRITE DUMMY END OF FILE AND EOF
          . TO UPDATE FILE
01754              WRITE 1,DUMMY
01760              WEOF 1
01762              CHAIN EXIT
01764              STOP

01765     START
01767     UPDAT
01771     ASK

01773     NAME
01775     ADDR
01777     SSN
02001     BUSNES
02003     CCODE
02005     TEMP
02007     UPDDTE
02011     END
02013     EXIT
02015     SLASH
02017     BUFF
02021     DUMMY
```

DATABUS COPY PROGRAM

```
        . PROGRAM COPY
        .
        . COPY FILE FROM FRONT DECK TO REAR OR REAR
        . DECK TO FRONT
        .
01000   EXIT    INIT "MASTER"
01011   BUFF    DIM 127
01213   TEMP    DIM 10
01230   FRONT   INIT "FRONT"
01240   BACK    INIT "BACK"
        .
01247   START   DISPLAY *H1,*V1,*EF,"COPY FRONT OR BACK TAPE?":
01307           KEYIN TEMP
01312           MATCH TEMP,FRONT
01315           GOTO COPYF IF EQUAL
01320           MATCH TEMP,BACK
01323           GOTO START IF NOT EQUAL
        .
        . COPY BACK DECK TO FRONT DECK
        .
01326   COPYB   REWIND 1
01330           PREPARE 2
01332           TRAP ENDB IF EOF1
01335   LOOPB   READ 1,BUFF
01341           WRITE 2,BUFF
01345           GOTO LOOPB
01347   ENDB    WEOF 2
01351           CHAIN EXIT
        .
        . COPY FRONT DECK TO BACK DECK
        .
01353   COPYF   REWIND 2
01355           PREPARE 1
01357           TRAP ENDF IF EOF2
01362   LOOPF   READ 2,BUFF
01366           WRITE 1,BUFF
01372           GOTO LOOPB
01374   ENDF    WEOF 1
01351           CHAIN EXIT
        .
        . COPY FRONT DECK TO BACK DECK
        .
01353   COPYF   REWIND 2
01355           PREPARE 1
01357           TRAP ENDF IF EOF2
01362   LOOPF   READ 2,BUFF
01366           WRITE 1,BUFF
01372           GOTO LOOPF
01374   ENDF    WEOF 1
01376           CHAIN EXIT
01400           STOP
```

DATABUS COPY PROGRAM

```
01401    START
01403    COPYF
01405    COPYB
01407    ENDB
01411    LOOPB
01413    ENDF
01415    LOOPF

01417    EXIT
01421    BUFF
01423    TEMP
01425    FRONT
01427    BACK
```

11.3 DATABUS 3

11.3.1 Instruction Summary

```
Directives
  DIM n
  INIT "character string"
  DIM *n
  INIT *"CHARACTER STRING"

Control
  TRAP (label) IF (event)
  GOTO (label)
  GOTO (label) IF (condition)
  GOTO (label) IF NOT (condition)
  RETURN
  RETURN IF (condition)
  RETURN IF NOT (condition)
  STOP
  STOP IF (condition)
  STOP IF NOT (condition)
  CHAIN (svar)
  BRANCH (index) OF (label list)

String
  CMATCH (sval) TO (sval)
  CMOVE (sval) TO (svar)
  MATCH (svar) TO (svar)
  MOVE (svar) TO (svar)
  APPEND (svar) TO (svar)
  RESET (svar)
  BUMP (svar)
  ENDSET (svar)
  EXTEND (svar)
  CLEAR (svar)
  LOAD (svar) FROM (index) OF (svar list)
  STORE (svar) INTO (index) OF (svar list)

Numeric Index Arithmetic
  ADD (index) TO (index)
  ADD n TO (index)
  SUB (index) FROM (index)
  SUB n FROM (index)
  COMPARE (index) TO (index)
  COMPARE n TO (index)
  MOVE (move) TO (svar)
  MOVE (svar) TO (index)

Keyboard, C.R.T., Printer I/O
  KEYIN (list)
  DISPLAY (list)
  PRINT (list)
  BEEP
```

CLICK
DSENSE
KSENSE

Cassette Tape I/O
READ (unit),(list)
WRITE (unit),(list)
REWIND (unit)
PREPARE (unit)
WEOF (unit)
BSPR (unit)
BSPF (unit)
ADVR (unit)
ADVF (unit)

Mag Tape I/O
READ (unit),(list)
WRITE (unit),(list)
REWIND (unit)
PREPARE (unit)
WEOF (unit)
BSPR (unit)
BSPF (unit)
ADVR (unit)
ADVF (unit)
ADVFW (unit)
PBOF (unit)
PEOF (unit)
ASCII
EBCDIC
BCD

Communications I/O
SEND (list)
RECEIVE n,(list)
WAIT n
DIAL (svar)
CONNECT
DSCNCT

11.3.2 Conditions

OVER
LESS
EQUAL
ZERO
EOS
PARITY
TIME
ERROR

11.3.3 Events

EOF1	
EOF2	
EOF3	
EOF4	
EOT2	
EOT2	
EOT3	
EOT4	
FORM1	Old Tape Format
FORM2	" " "
FORM3	" " "
FORM4	" " "
RFAIL1	New Tape Format
RFAIL2	" " "
RFAIL3	" " "
RFAIL4	" " "
CFAIL	
RING	

11.3.4 User Area

<u>Interpreter Machine</u>

8K - 3000_8 bytes (14400_8-17400_8)
12K - 10000_8 bytes (20000_8-27777_8)
16K - 20000_8 bytes (20000_8-37777_8)

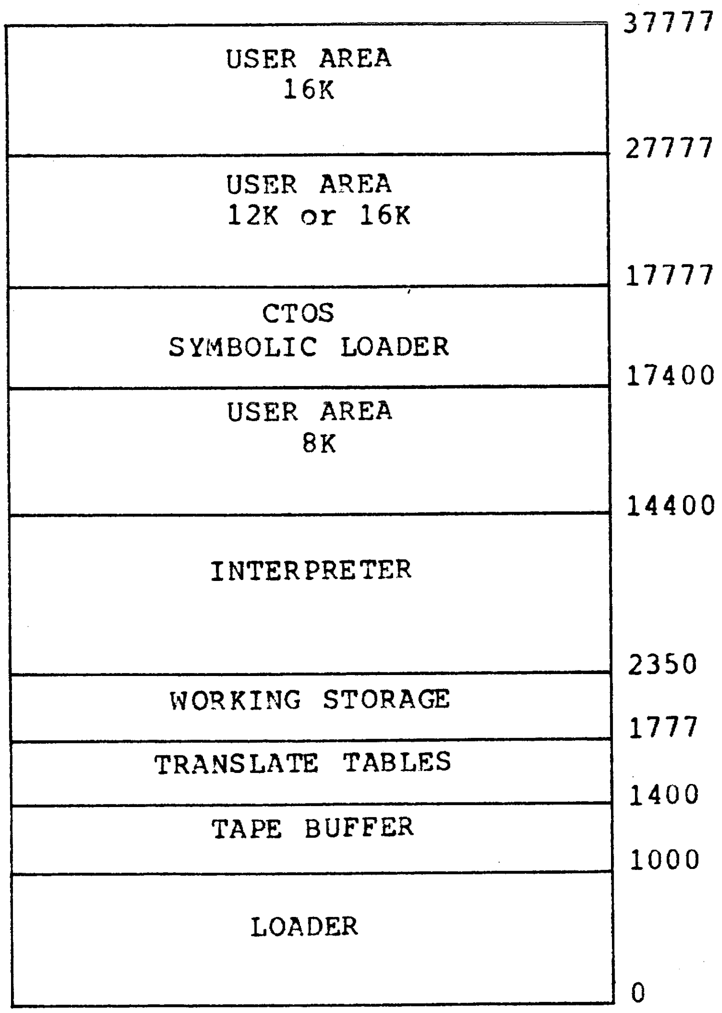
11.3.5 Dictionaries

<u>Compiler Machine</u>

8K - 100 labels, 100 variables
12K or 16K - 125 labels, 125 variables

11.3.6 Interpreter Internal Structure

Databus 3 is layed out in memory as follows:



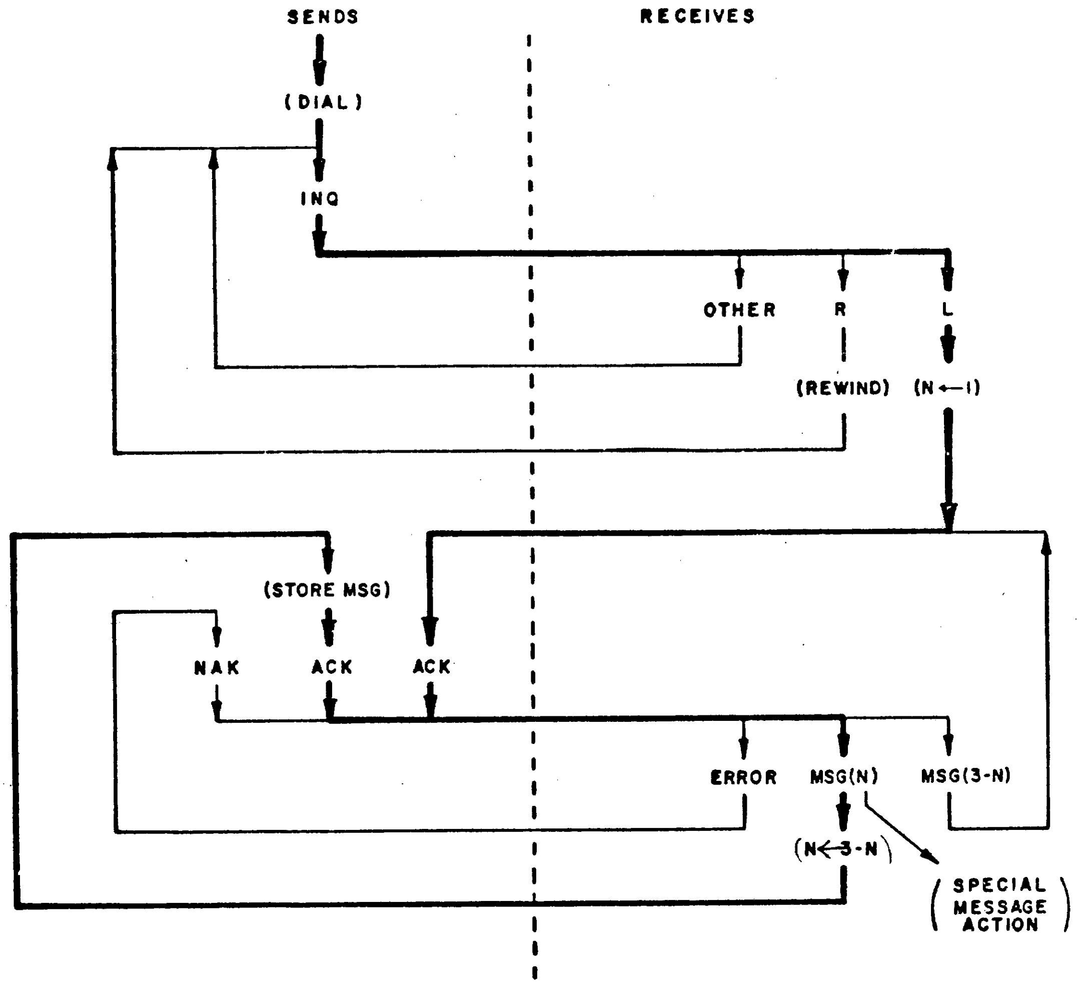
11.3.7 SAMPLE PROGRAMS

The sample DATABUS 3 programs included demonstrate a MASTER-SLAVE communications system with logically complete error control. Use of a serial number (modulo two) insures that no message will be lost or repeated. One can completely lose the connection (e.g. telephone disconnects) and subsequently restore it to continue data transfer without losing a bit.

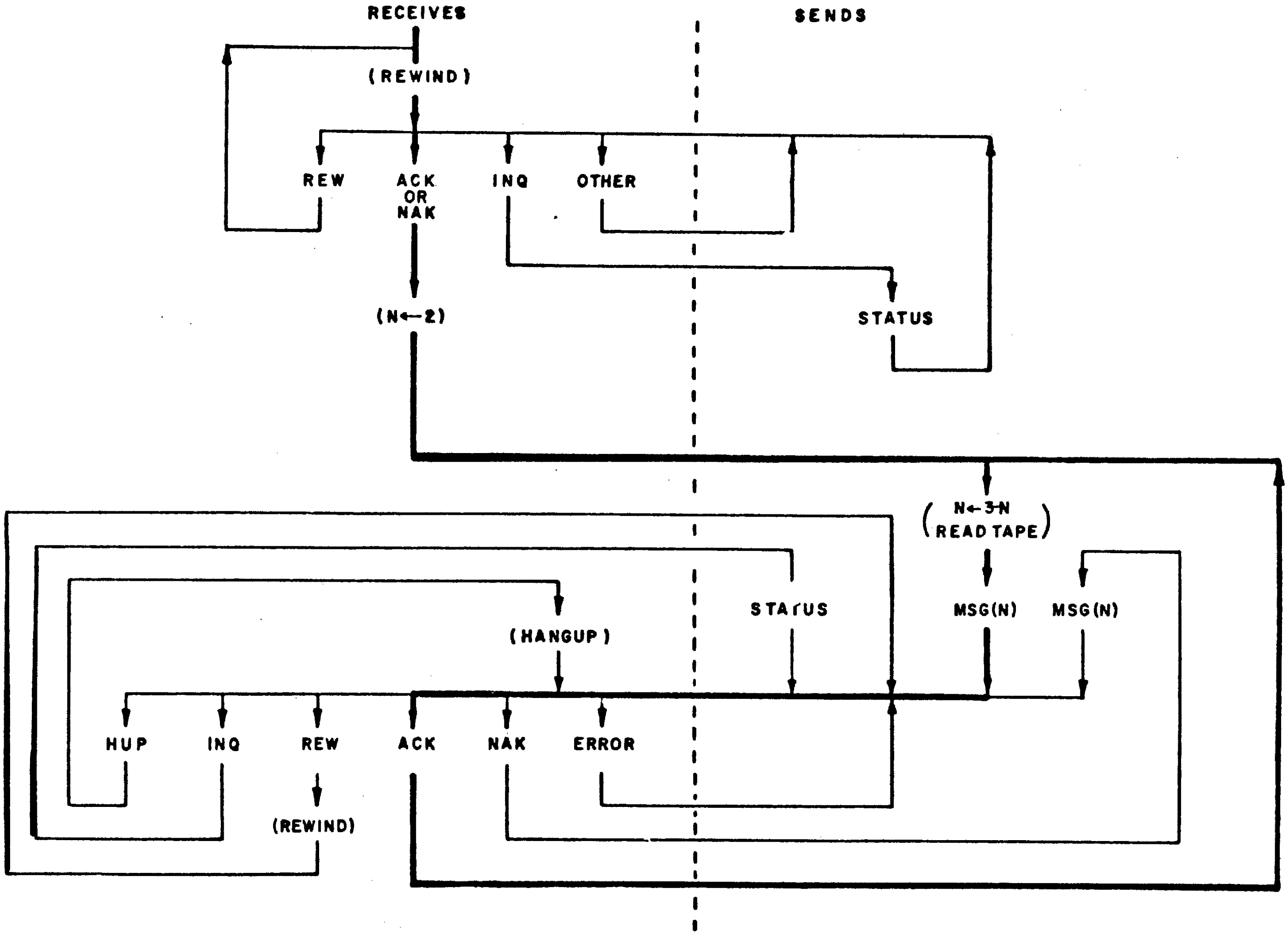
The master station dials the slave station to obtain a tape file of information. The master program displays the information it receives but could be modified to print it or write it to tape. The MASTER-SLAVE station discipline is designed to be easily modified for use in a multi-drop environment using dedicated communications lines. The contents of the ADR variable of each slave station (DATABUS 6 simulator) would be unique and the master station would address a particular station by setting the value of its ADR variable to correspond to the address of the desired slave station.

Message flow diagrams are included to clarify the functions of the programs. The heavy lines indicate the normal sequence of operations while the other lines indicate paths taken to handle special conditions invoked by errors in transmission. Capitalized items indicate messages sent while parenthesized items indicate actions taken.

MASTER STATION (MESSAGE RECEIVED) DISCIPLINE:



SLAVE STATION (MESSAGE TRANSMITTER) DISCIPLINE:



NOTES: 1. ANSWER IF RINGING DETECTED WHILE WAITING FOR A MESSAGE
2. HANG UP IF A VALID MESSAGE IS NOT RECEIVED FOR 45 SECONDS

DATABUS 3 MASTER STATION FOR DATABUS 6

```
        .
        . TESTS DATABUS 6 COMMUNICATIONS WITH DATABUS 3
        .
14400   INQ      INIT 5
14404   REW      INIT 010
14410   HUP      INIT 033
14414   ACK      INIT 6
14420   EOF      INIT 020
14430   PARITY   INIT 021
14434   EOT      INIT 022
        .
14440   NUMBER   DIM 15
14462   MSG      DIM 100
14631   ADR      INIT "A"
14635   TWO      INIT "2"
14641   ONE      INIT "1"
14645   ALT      INIT "1"
14651   RXALT    INIT "0"
        .
14655            CALL DIAL
14657            GOTO START
14661   REWIND   SEND REW,ADR
14665   START    SEND INQ,ADR
14671            RECEIVE 2,MSG
14675            GOTO STAR IF ERROR
14700            DISPLAY *H1,*V12,"STATUS:",MSG
14717            CMATCH MSG,"R"
14722            GOTO REWIND IF EQUAL
14725            CMATCH MSG,"N"
14730            GOTO REWIND IF EQUAL
14733            CMATCH MSG,"L"
14736            GOTO START IF NOT EQUAL
        .
14741   SACK     SEND ACK,ADR
14745            GOTO GETHUP
14747   SNAK     SEND NAK,ADR
14753   GETHUP   DSENSE
14754            GOTO GETMSG IF NOT EQUAL
14757            RECEIVE 2,RXALT,MSG
14764            SEND HUP,ADDR
14770            CALL DIAL1
14772   GETMSG   RECEIVE 2,RXALT,MSG
14777            GOTO SNAK IF ERROR
15002            CMATCH RXALT,ALT
15005            GOTO SACK IF NOT EQUAL
15010            CMATCH PARITY TO MSG
15013            GOTO PFAIL IF EQUAL
15016            CMATCH EOF TO MSG
15021            GOTO DONE IF EQUAL
15024            CMATCH EOT TO MSG
15027            GOTO EOTM IF EQUAL
15032            DISPLAY *H1,*V12,MSG
```

```
15041  FLIP     MATCH ALT,ONE
15044           GOTO ALTWO IF EQUAL
15047           MOVE ONE,ALT
15052           GOTO TFW
15054  ALTWO    MOVE TWO,ALT
15057  TFW      KSENSE
15060           GOTO REWIND IF EQUAL
15063           GOTO SACK
15065  PFAIL    DISPLAY *H1,*V12,"*** PARITY ERROR ON TAPE***"
15127           BEEP
15130           GOTO FLIP
15132  EOTM     DISPLAY *H1,*V12,"*** END OF TAPE ***"
15163  DONE     DISPLAY *H1,*V12,*R,"END OF TRANSACTION"
15214           SEND HUP,ADR
15220           DSCNCT
15221           STOP
       .
15222  DIAL     DSCNCT
15223           BEEP
15224           KEYIN *H1,*V12,"PHONE NUMBER:",NUMBER
15251  DIAL1    DISPLAY *H1,*V12,*R,"I'M DIALING",*R
15274           DIAL NUMBER
15276           CONNECT
15277           SUB I1,I1
15302  DIAL2    ADD 1 TO I1
15305           SEND INQ,ADR
15311           RECEIVE 2,MSG
15315           RETURN IF NOT ERROR
15317           COMPARE 10 TO I1
15322           GOTO DIAL2 IF LESS
15325           KSENSE
15326           GOTO DIAL IF EQUAL
15331           GOTO DIAL1
15333           STOP

15334  DIAL
15336  START
15340  REWIND
15342  SACK
15346  SNAK
15350  GETMSG
15352  DIAL1
15354  PFAIL
15356  DONE
15360  EOTM
15362  FLIP
15364  ALTWO
15366  TFW
15370  DIAL2

15372  INQ
15374  REW
15376  HUP
```

DATABUS 3 MASTER STATION FOR DATABUS 6

15400	ACK
15402	NAK
15404	EOF
15406	PARITY
15410	EOT
15412	NUMBER
15414	MSG
15416	ADR
15420	TWO
15422	ONE
15424	ALT
15426	RXALT

DATABUS 6 SIMULATOR

```
        . PROGRAM DATABUS 6 SIMULATOR
        .
        . SIMULATES DATABUS 6 SEND FUNCTION WITH DATABUS 3
        .
14400   INQ         INIT 5
14404   REW         INIT 010
14410   HUP         INIT 033
14414   ACK         INIT 6
14423   NAK         INIT 025
14424   EOF         INIT 020
14430   RDY         INIT "R"
14434   LP          INIT "L"
14440   ADR         INIT "A"
14444   ONE         INIT "1"
14450   TWO         INIT "2"
        .
14454   STATUS      DIM 1
14460   MSG         DIM 100
14627   RESPM       DIM 10
14644   ADRR        DIM 1
14650   MSGNR       DIM 1
        .
14654               TRAP EOF IF EOF2
14657               TRAP ANSWER IF RING
14662   REWIND      CALL REWND2
14664               MOVE TWO TO MSGNR
14667   GOWAIT      RECEIVE 45,RESPM,ADRR
14674               CALL HANGUP IF TIME
14677               GOTO GOWAIT IF ERROR
14702               CMATCH ADR TO ADRR
14705               GOTO GOWAIT IF NOT EQUAL
14710               CMATCH ACK TO RESPM
14713               GOTO GETREC IF EQUAL
14716               CMATCH NAK TO RESPM
14721               GOTO GETREC IF EQUAL
14724               CMATCH REW TO RESPM
14727               CALL REWND2 IF EQUAL
14732               CMATCH INQ TO RESPM
14735               CALL SSTAT IF EQUAL
14740               GOTO GOWAIT
        .
14742   GETREC      MOVE RDY TO STATUS
14745               READ 2,MSG
14751   FLIP        MATCH ONE TO MSGNR
14754               GOTO FLIP2 IF EQUAL
14757               MOVE ONE TO MSGNR
14762               GOTO SNDREC
14764   FLIP2       MOVE TWO TO MSGNR
        .
14767   SNDREC      SEND MSGNR,MSG
14773   GETRSP      RECEIVE 45,RESPM,ADRR
15000               CALL HANGUP IF TIME
```

```
15003                  GOTO GETRSP IF ERROR
15006                  CMATCH ADR TO ADRR
15011                  GOTO GETRSP IF NOT EQUAL
15014                  CMATCH ACK TO RESPM
15017                  GOTO GETREC IF EQUAL
15022                  CMATCH NAK TO RESPM
15025                  GOTO SNDDREC IF EQUAL
15030                  CMATCH REW TO RESPM
15033                  CALL REWND2 IF EQUAL
15036                  CMATCH HUP TO RESPM
15041                  CALL HANGUP IF EQUAL
15044                  CMATCH INQ TO RESPM
15047                  CALL SSTAT IF EQUAL
15052                  GOTO GETRSP
       .
15054  HANGUP          DSCNCT
15055                  RETURN
15056  ANSWER          CONNECT
15057                  RETURN
15060  REWND2          REWIND 2
15062                  MOVE LP TO STATUS
15065                  RETURN
15066  SSTAT           SEND STATUS
15071                  RETURN
15072  EOF             MOVE EOF TO MSG
15075                  GOTO FLIP
15077                  STOP

15100  EOF
15102  ANSWER
15104  REWND2
15106  REWIND
15110  GOWAIT
15112  HANGUP
15114  GETREC
15116  SSTAT
15120  FLIP
15122  FLIP2
15124  SNDREC
15126  GETRSP

15130  INQ
15132  REW
15134  HUP
15136  ACK
15140  NAK
15142  EOF
15144  RDY
15146  LP
15150  ADR
15152  ONE
15154  TWO
15156  STATUS
```

DATABUS 6 SIMULATOR

15160	MSG
15162	RESPM
15164	ADRR
15166	MSGNR

11.4 DATABUS 4

11.4.1 Instruction Summary

Directives

```
DIM n
INIT "character string"
DIM *n
INIT *"CHARACTER STRING"
LENGTH (size)
```

Control

```
TRAP (label) IF (event)
GOTO (label)
GOTO (label) IF (condition)
GOTO (label) IF NOT (condition)
CALL (label)
CALL (label) IF (condition)
CALL (label) IF NOT (condition)
RETURN
RETURN IF (condition)
RETURN IF NOT (condition)
STOP
STOP IF (condition)
STOP IF NOT (condition)
CHAIN (svar)
```

String

```
MATCH (svar) TO (svar)
CMATCH (literal),(svar),n
RANGE (svar),(literal),(literal)
```

Numeric Index Arithmetic

```
ADD (index) TO (index)
ADD n TO (index)
SUB (index) FROM (index)
SUB n FROM (index)
COMPARE (index) TO (index)
COMPARE n TO (index)
```

Keyboard, C.R.T., Printer I/O

```
KEYIN (list)
DISPLAY (list)
PRINT (list)
BEEP
CLICK
DSENSE
KSENSE
```

Cassette Tape I/O

```
READ (unit),(list)
WRITE (unit),(list)
REWIND (unit)
```

PREPARE (unit)
BKSP (unit)
WEOF (unit)

11.4.2 Conditions

LESS
EQUAL
ZERO
EOS

11.4.3 Events

EOF1	
EOF2	
EOT1	
EOT2	
FORM1	Old Tape Format
FORM2	" " "
RFAIL1	New Tape Format
RFAIL2	" " "
CFAIL	

11.4.4 User Area

<u>Interpreter Machine</u>

4K - 1400_8 bytes (6400_8-7777_8)
6K - 5400_8 bytes (6400_8-13777_8)
8K - 11400_8 bytes (6400_8-17777_8)
12K - 15400_8 bytes (6400_8-27777_8)
16K - 21400_8 bytes (6400_8-37777_8)

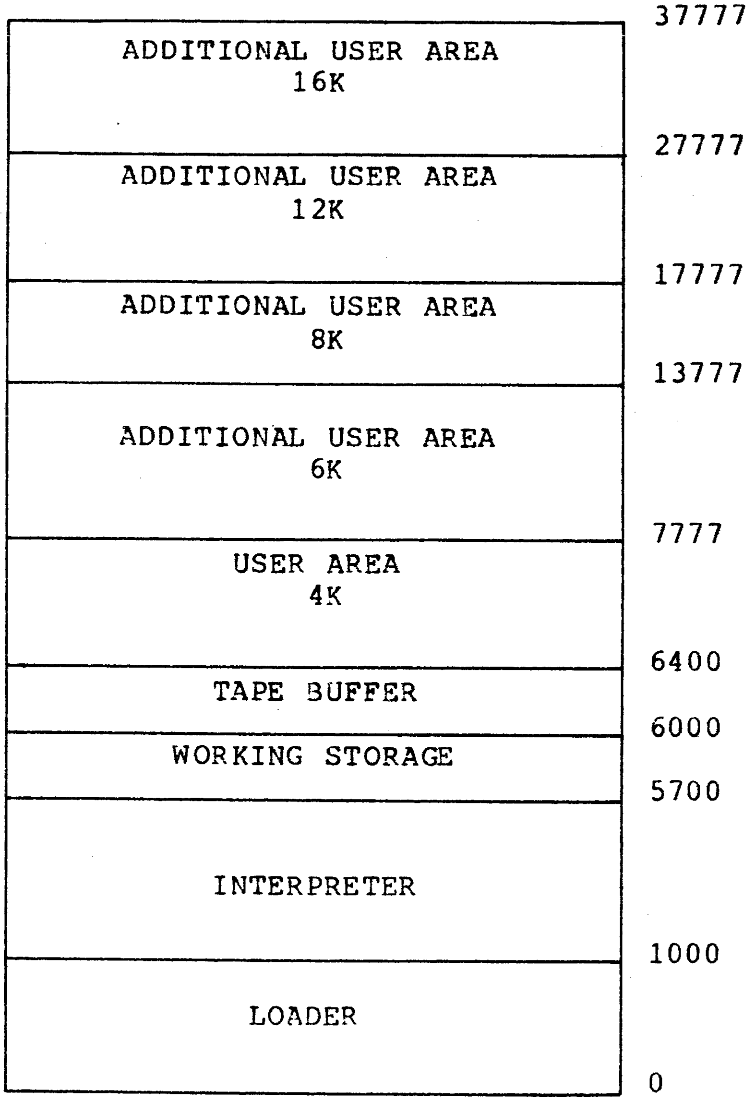
11.4.5 Dictionaries

<u>Compiler Machine</u>

8K - 100 labels, 100 variables
12K or 16K - 125 labels, 125 variables

11.4.6 Interpreter Internal Structure

Databus 4 is layed out in memory as follows:



11.4.7 Sample Program

```
      . PROGRAM PAYROLL DATA ENTRY PROGRAM
      .
06400 ENUM    DIM 5
06410 ENAM    DIM 25
06444 TITL    DIM 3
06452 DEPT    DIM 3
06460 DEPN    DIM 2
06465 SSN     DIM 11
06503 EN      DIM 1
06507 SEX     DIM 1
06513 DATE    DIM 6
06524 HOUR    DIM 6
06535 LSI     DIM 8
06550 DLSI    DIM 6
06561 BIRTH   DIM 6
06572 STAX    DIM 6
06603 CTAX    DIM 6
06614 INS     DIM 6
06625 FICNE   DIM 1
06631 RESP    DIM 3
      .
06637 START   KEYIN *H1,*V3,*EF,*H8,"PAYROLL DATA ENTRY":
06677         *45,"DATE(MMDDY):"DATE
06714         RANGE DATE,060,071
06720         GOTO START IF NOT EQUAL
06723         PREPARE 2
06725 INPUT   DISPLAY *H1,*V3,*EF,"EMP #:",*H1,*V4:
06745         "EMP NAME:",*H1,*V5,"TITLE:",*H1,*V6,"DEPT:":
07001         *H1,*V7,"# DEPN:",*H1,*V8,"SSN:":
07024         *H1,*V9,"N/E:",*H1,*V10,"SEX:"


      .
07045 INUM    KEYIN *H8,*V3,*EL,ENUM
07055         RANGE ENUM,060,071
07061         GOTO INUM IF NOT EQUAL
07064         KEYIN *H11,*V4,*EL,ENAM
07074 ITITL   KEYIN *H8,*V5,*EL,TITL
07104         RANGE TITL,060,071
07110         GOTO ITITL IF NOT EQUAL
07113         KEYIN *H7,*V6,*EL,DEPT
07123 IDEPN   KEYIN *H9,*V7,*EL,DEPN
07132         RANGE DEPN,060,071
07137         GOTO IDEPN IF NOT EQUAL
07142 ISSN    KEYIN *H6,*V8,*EL,SSN
07152         RANGE SSN,055,071
07156         GOTO IDEPN IF NOT EQUAL
07161 IEN     KEYIN *H6,*V9,*EL,EN
07171         CMATCH "E",EN
07175         GOTO IMF IF EQUAL
07200         CMATCH "N",EN
```

DATABUS 4 PAYROLL DATA ENTRY PROGRAM

```
07204           GOTO IEN IF NOT EQUAL
07207   IMF     KEYIN *H6,*V10,*EL,SEX
07217           CMATCH "M",SEX,1
07223           GOTO INPT IF EQUAL
07226           CMATCH "F",SEX,1
07232           GOTO IMF IF NOT EQUAL
        .
07235   INPT    DISPLAY *H40,*V3,"RATE/HR:",*H40,*V4:
07256           "LAST INCR:",*H40,*V5,"DATE LAST ":
07306           "INCR:",*H40,*V6,"BIRTH:",*H40,*V7:
07331           "STATE TX:",*H40,*V8,"CITY TX:",*H40:
07360           *V9,"INS:",*H40,*V10,"FICA N/E:"
        .
07404   IHOUR   KEYIN *H49,*V3,*EL,HOUR
07414           RANGE HOUR,056,071
07420           GOTO IHOUR IF NOT EQUAL
07423   ILSI    KEYIN *H51,*V4,*EL,LSI
07433           RANGE LSI,056,071
07437           GOTO ILSI IF NOT EQUAL
07442   IDLSI   KEYIN *H47,*V6,*EL,BIRTH
07452           RANGE DLSI,060,071
07456           GOTO IDLSI IF NOT EQUAL
07461   IBRTH   KEYIN *H47,*V6,*EL,BIRTH
07471           RANGE BIRTH,060,071
07475           GOTO IBRTH IF NOT EQUAL
07500           KEYIN *H50,*V7,*EL,STAX
07510           KEYIN *H49,*V8,*EL,CTAX
07520           KEYIN *H45,*V9,*EL,INS
07530   IFIC    KEYIN *H50,*V10,*EL,FICNE
07540           CMATCH "N",FICNE,1
07542           GOTO IRSP IF EQUAL
07547           CMATCH "E",FICNE,1
07553           GOTO IFIC IF NOT EQUAL
        .
07556   IRSP    KEYIN *H45,*V12,"CORRECT?",RESP:
07575           CMATCH "N",RESP
07601           GOTO INPUT IF EQUAL
07604           CMATCH "Y",RESP
07610           GOTO IRSP IF NOT EQUAL
        .
07613           WRITE 2,ENUM,ENAM,TITL,DEPT,DEPN,SSN,EN,SEX:
07625           HOUR,LSI,DLSI,BIRTH,STAX,CTAX,INS,FINCE
07636           WEOF 2
        .
07640   ASK     KEYIN *H45,*V12,*EL,"CONT?",RESP:
07650           CMATCH "Y",RESP
07661           GOTO INPUT IF EQUAL
07664           CMATCH "N",RESP
07670           GOTO ASK IF NOT EQUAL
07673           STOP

07674   START
07676   INPUT
```

DATABUS 4 PAYROLL DATA ENTRY PROGRAM

```
07700   INUM
07702   ITITL
07704   IDEPN
07706   ISSN
07710   IEN
07712   IMF
07714   INPT
07716   IHOUR
07720   ILSI
07722   IDLSI
07724   IBRTH
07726   IFIC
07730   IRSP
07732   ASK

07734   ENUM
07736   ENAM
07740   TITL
07742   DEPT
07744   DEPN
07746   SSN
07750   EN
07752   SEX
07754   DATE
07756   HOUR
07760   LSI
07762   DLSI
07764   BIRTH
07766   STAX
07770   CTAX
07772   INS
07774   FICNE
07776   RESP
```

11.5 DATABUS 5

11.5.1 Instruction Summary

```
Directives
  DIM n
  INIT "character string"
  DIM *n
  INIT "CHARACTER STRING"
  LENGTH (size)

Control
  TRAP (label) IF (event)
  GOTO (label)
  GOTO (label) IF (condition)
  GOTO (label) IF NOT (condition)
  CALL (label)
  CALL (label) IF (condition)
  CALL (label) IF NOT (condition)
  RETURN
  RETURN IF (condition)
  RETURN IF NOT (condition)
  STOP
  STOP IF (condition)
  STOP IF NOT (condition)
  CHAIN (svar)
  ACALL (address)

String
  MATCH (svar) TO (svar)
  CMATCH (literal),(svar),n
  RANGE (svar),(literal),(literal)

Numeric Index Arithmetic
  ADD (index) TO (index)
  ADD n TO (index)
  SUB (index) FROM (index)
  SUB n FROM (index)
  COMPARE n TO (index)
  MOVE (index) TO (svar)
  MOVE (svar) TO (index)

Keyboard, CRT, Printer I/O
  KEYIN (list)
  DISPLAY (list)
  PRINT (list)
  BEEP
  CLICK
  DSENSE
  KSENSE

Cassette Tape I/O
  READ (unit),(list)
```

```
WRITE (unit),(list)
REWIND (unit)
PREPARE (unit)
BKSP (unit)
WEOF (unit)
```

11.5.2 Conditions

LESS
EQUAL
ZERO
EOS

11.5.3 Events

EOF1	
EOF2	
EOT1	
EOT2	
FORM1	Old Tape Format
FORM2	" " "
RFAIL1	New Tape Format
RFAIL2	" " "
CFAIL	

11.5.4 User Area

<u>Interpreter Machine</u>

4K - 1100_8 bytes (6700_8-7777_8)
6K - 5100_8 bytes (6700_8-13777_8)
8K - 11100_8 bytes (6700_8-17777_8)
12K - 15100_8 bytes (6700_8-27777_8)
16K - 21100_8 bytes (6700_8-37777_8)

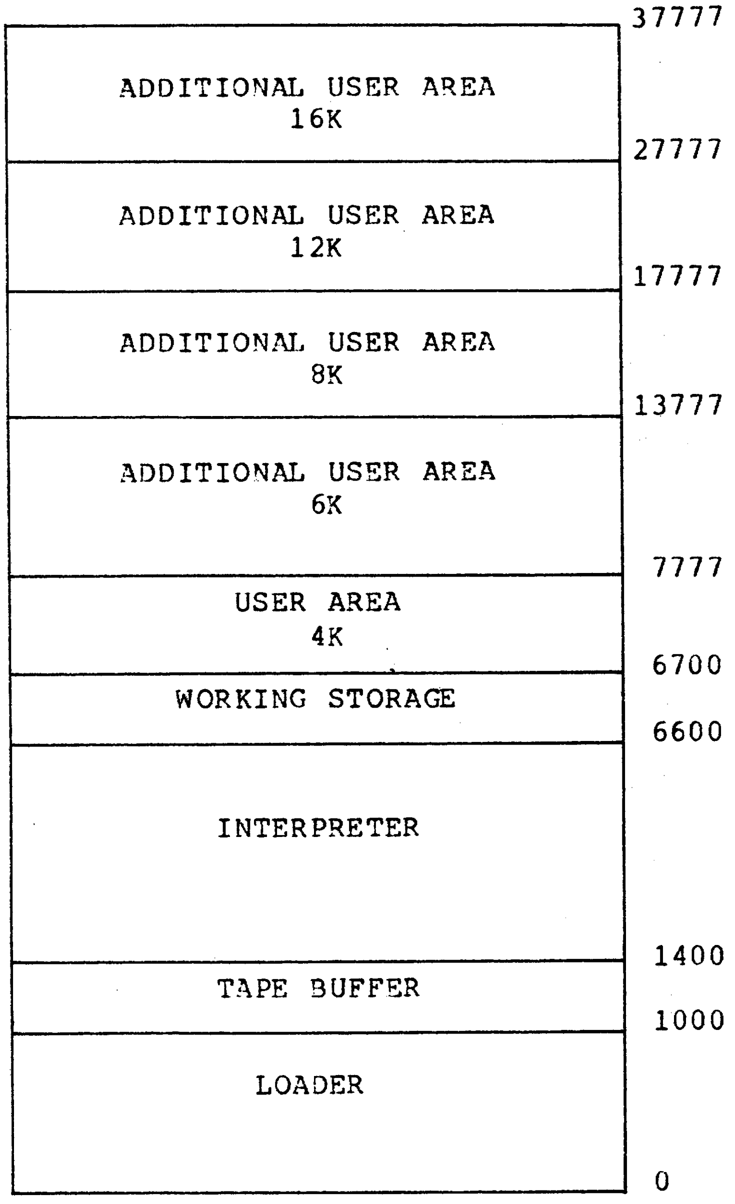
11.5.5 Dictionaries

<u>Compiler Machine</u>

8K - 125 labels, 125 variables
12K or 16K - 125 labels, 125 variables

11.5.6 Interpreter Internal Structure

Databus 5 is layed out in memory as follows:



11.5.7 Sample Programs

See the Databus 4 Sample Program.

```
PAGE 1  SAMPLE ASSEMBLY PROGRAM FOR ACALL INSTRUCTION

402 LABELS LEFT

LABELS NOT USED WERE:           ENTRY

      03534                     DSP$
      14003                     ENTRY
      14020                     MESG
      14000                     SUBR
```

```
PAGE 2  SAMPLE ASSEMBLY PROGRAM FOR ACALL INSTRUCTION

                                            .
                                            .ASSEMBLY PROGRAM FOR DATABUS 5 CALL
                                            .
14000                                             SET   014000
                                            .
14000  104  000  003                        SUBR  JMP   01400  FIRST STATEMENT MUST
                                            .                  BE JUMP BACK TO
                                            .                  INTERPRETER ENTRY POINT
14003  020                                  ENTRY BEEP         THIS IS THE ASSEMBLY
                                            .                  SUBPROGRAM ENTRY POINT
14004  066  020  056  030                         HL    MESG
14010  036  050                                   LD    40
14012  046  013                                   LE    11
14014  106  134  007                              CALL  DSP$
14017  007                                        RET          RETURN TO DATABUS INTER-
                                            .                  PRETER AT END OF SUB-
                                            .                  PROGRAM
03534                                       DSP$  EQU   03534  DISPLAY ROUTINE IN
                                            .                  INTERPRETER
                                            .
14020  101  103  101  114  114              MESG  DC    'ACALL TEST MESSAGE',0203
14025  040  124  105  123  124
14032  040  115  105  123  123
14037  101  107  105  203
                                            .
14000                                             END   SUBR
```

12.0 DATABUS 6

12.1 Introduction

DATABUS 6 is a system of programs designed to perform data capture functions. It consists of an operating system, which is run subsequent to depression of the RESTART key, and a set of programs which perform the various functions.

12.2 Global Features

The PUNCH, APPEND, VERIFY, and EDIT programs run with a similar appearance and use. A card column counter appears in the middle of the top line of the screen and indicates in which column the next character will be entered. There are two lines of significant data on the screen. The bottom line displayed will be referred to as the "punch station" and the one above it as the "read station" because of the closeness of their functional analogy to a conventional keypunch. All data is entered in the "punch station" which is transferred to the "read station" when the end of the line is reached. Data that is rolled out of the "read station" is written on tape unless the DISPLAY key is depressed at the time when the data is rolled out, in which case it is discarded. This latter action is similar to a conventional keypunch operator reaching up with his left hand and removing the card as it is rolled up into the stacker. Formatting of the input data is controlled by a program control card in a manner similar to a conventional keypunch. Control cards are generated through the use of the PROGRAM command and are stored on the system tape. Up to ten control cards may be kept on the system at one time and all may be displayed by issuing the DISPLAY command. Any one of the ten control cards may be selected for use at any point by the PUNCH, APPEND, and VERIFY programs.

Control functions in PUNCH, APPEND, VERIFY, and EDIT are achieved using the SHIFT key in conjunction with certain letter keys. The letter used usually has mnemonic value (R for release, D for duplicate, etc.) for easy recollection by the operator. The shift key used on the characters UIOJKLM,.P will produce the digits 1234567890 respectively in emulation of a conventional keypunch numeric pad. Some other keys will produce functions as denoted in the following table. Any keys not mentioned above or below will be ignored with a beep, letting the operator know he struck an invalid key. The function of the following control keys will become clear in the descriptions of the various programs. If program names appear in parenthesis after a control key description, it is implied that the control key has validity only in those programs. The EDIT program has an implied program control card consisting of a field delimiter in every zero modulo ten column (10,20,...,80).

Control Characters

R - Release the card.

D - Duplicate the next column only.

C - Copy to the next field delimiter on the program control card.

W - Copy the whole card from the current column on.

Q - Quit (ignored unless the column counter equals 01).

X - Waits for another character to be entered from the keyboard. This must be a shifted "E". Entering any other character causes return to normal mode. Entering an E will cause the program to quit without reading the rest of the data from the front tape. This is useful if the front tape has no end of file mark or is deviant from the normal format in some other manner.(EDIT)

S - Search for a record that matches the search key (ignored unless the column counter equals 01). (EDIT)

G - Get the next data record from the front deck (ignored unless the column counter equals 01).(EDIT)

N - Turn on the program control card control. Program control ON is the mode assumed when the program is started. (PUNCH, APPEND, VERIFY)

F - Turn off the program control card control.(PUNCH, APPEND, VERIFY)

V - Waits for another character to be entered from the keyboard. This must be a digit or a space. Entering a space returns to the normal entry mode (an escape from program selection), whereas, entering some digit will cause the corresponding program control card image to be read from the system tape and used as the program control card. (PUNCH, APPEND, VERIFY) Note that if the PROGRAM command has never been given for a specific control card, the card will be empty (no field delimiters). When the PUNCH, APPEND, or VERIFY program is started, a program control card of all field delimiters is assumed.

Z - Allow correction in the following field. (VERIFY)

In addition to the above, the ENTER and ; keys perform the SKIP function of the conventional keypunch. The BSP key will backspace one column unless the column counter is 01, in which case a beep is sounded. The CANCEL key will backspace until either the beginning of the card or a field delimiter in the program control card is reached. One may enter a semicolon by striking the ′ key (lower case to the right of the P key), the < character by striking a {, and the > character by striking the }.

12.3 Functional Descriptions

The DATABUS 6 operating system has a command interpreter with syntax rules similiar to CTOS. There are nine commands that may be issued. Entering an illegal command will cause a response of "What?" after which a valid command should be issued.

PUNCH allows data to be entered directly on the front cassette. The operating system will ask "New tape in front deck?". At this point or before, the operator should place in the front deck of the machine a tape upon which there is no valuable information. He should then depress the Y followed by the ENTER key. Depressing N instead of Y will return control to the operating system. When the column counter appears on the screen, the program is ready for data entry. If, upon entering many records, the physical end of the front cassette is reached, a logical end of file mark will be written over the last record written, the front tape will be rewound, and control will be returned to the operating system. To terminate the PUNCH operation, the operator issues the Q command which will write the record resting in the read station, follow it with a logical end of file mark, rewind the front cassette and return control to the operating system.

APPEND performs the same function as PUNCH except the operating system will ask "Old tape in front deck?". At this point or before, the operator should place in the front deck of the machine a tape upon which data has previously been entered. He should then depress the Y followed by the ENTER key. The operating system will position the front tape after the last data record and then pass control to the PUNCH program. This function allows the operator to append more records to a tape already containing data.

VERIFY allows the operator to verify information on a data tape in the conventional manner. Upon issuance of the command, the operating system will ask "Old tape in front deck?". At this point or before, the operator should place in the front deck of the machine a tape upon which data has been previously entered. He should then depress the Y followed by the ENTER key. The operating system will position the front tape to the first data record. The first data record is then read and displayed in the "read station". The operator then enters the same line from the keyboard. Any discrepancy with the line obtained from the front tape will be greeted with a beep and rejection of the character from the keyboard will occur. Correction of a field will be allowed if the Z command is issued. If a character is changed, a beep will be sounded. Changing characters will once again be disallowed upon entering the next field. When the end of the line is reached, the line entered will be compared to the record obtained from the front deck. If a change is detected, the record on the front tape will be overwritten to reflect the change. The next record is then read from the front tape and the process is repeated until the end of the front tape is reached (logical or physical) or the Q command is issued. (Note that two successive VERIFY's must not be performed without an intervening EDIT for physical record realignment purposes.)

EDIT allows corrrection, addition, and deletion of records that are on tape already containing data. Operator action is similar to that required for the APPEND function until the program begins to run. The program will position the front tape to the first data record and the rear tape to a scratch area. As the operator goes through the records, they will be read from the front tape and written on the rear tape. If the end of data is found on the front tape (logical or physical end of file), a blank line will be assumed for the data. When the operation is concluded with the Q command, the program will make sure that all of the data has been copied from the front tape to the rear tape (if the X command is given, the rest of the data on the front tape is discarded) and then rewind both tapes back to the first record and copy the rear tape back to the front tape. If it is desired to have the updated data put on a fresh cassette, the operator may remove the old data tape from the front deck and insert a new tape (need not be rewound or prepared in any way) while the program is rewinding the rear tape (performed with a slew causing a considerable delay with long files).

The EDIT program uses a search key when it is desired to search down the tape for a certain record. The key is entered as a normal data line except that two characters have particular significance. The underline character (to the right of the equals sign) will cause the corresponding column in the record obtained from the front tape to be assumed to match the key. The vertical bar (shifted key to the right of the P key) will cause the corresponding column and all that follows in the record obtained from the front tape to be assumed to match the key. After the search key is entered, it will be resting in the "read station". At this point the operator issues the S command (can be issued only when the column counter is equal to 01) and the EDIT program reads the front tape looking for a record that matches the search key. If a record does not match, it is written on the rear tape and the next record is read from the front tape. When an unrecoverable parity error (in which case the first column is set to a percent sign) or the desired record is found, it will be displayed (in the case of a parity error, whatever was read will be displayed and the tape will be positioned after the faulty record) and left resting in the "read station" with the search key being discarded. If the logical or physical end of file is reached, the tape will be left sitting before the end of file marker and a blank line assumed for the data. Note that the G command is equivalent to entering a search key consisting of a vertical bar in the first column. At this point the record may be corrected. If it is desired that it be discarded, the operator must depress the DISPLAY key when the record is rolled out of the "read station". Lines may be inserted at this point by simply entering them. Lines may be deleted by issuing the G command while depressing the DISPLAY key.

Note that if the physical end of the rear tape is reached while the edit is being performed, control will be returned to the operating system. If the operator wishes to recover the data on the rear tape, he may use the DUPLICATE function.

DUPLICATE allows the operator to transfer the data from the scratch area on the rear tape to a tape in the front deck. If the end of tape is reached on the rear deck, it will appear identically to a logical end of file mark. If the end of tape is reached on the front deck, control will be returned to the operating system.

PROGRAM allows the operator to create up to ten different program control cards for use in the PUNCH, APPEND, and VERIFY programs. Upon entering the PROGRAM command, the operating system will ask for a program number. Numbers allowed are the digits 0 through 9. Note that characters following the first will be ignored. The digit entered will correspond to the one used following the V command in the PUNCH, APPEND and VERIFY programs. If it is decided that a new program control card entry is not desired, just striking the ENTER key for the program number will cause control to be returned to the operating system. After receiving a valid digit, the PROGRAM function will position the system tape to the location of the particular program control card image involved and then display a form on the screen which allows the operator to see in which column he is entering his control information. At this point, six entries are allowed:

SPACE - no control information
F - field delimiter
D - auto-duplicate
S - auto-skip
BSP - erase previous character entered
ENTER - end of card entry

F, D, and S have the same meaning as for a conventional card punch but note that only one may go in any particular column. At any point in entering the control information, depression of the ENTER key will cause any following columns to be assumed as spaces and the card to be written on the system tape. Control will then be returned to the operating system.

DISPLAY displays the contents of all the program control cards on the screen. Two formatting lines are written on the screen to enable the operator to determine in which columns the characters reside. Card zero is displayed as the first line, one as the second, and so forth to card nine as the last.

SEND sends the data contained on the tape in the front deck over a communications line using DATABUS 3 discipline. Either a direct connection or the switched network may be used for this - and the system is capable of muti-drop operation on direct connections. A sample DATABUS 3 program is included which shows the coding necessary to communicate with the DATABUS 6 program. Since the SEND program accepts DATABUS format tapes, any tape generated by any DATABUS may be sent by it. Note that numeric items on the tape will be converted to strings in the transmission process. Error control is logically complete (e.g., the telephone connection can be lost and restored without losing any information) and either point-to-point or multi-drop operating procedures can be used. The SEND program has an address for multi-drop purposes which can be changed using the ADDRESS function in the DATABUS 6 operating system. When the SEND command is given, the operating system will ask "Old tape in front deck?". At this point or before, the operator should place in the front deck of the machine a tape upon which data has previously been entered. He should then depress the Y followed by the ENTER key. The operating system will position the tape to the first data record and then pass control to the SEND program. The SEND program will wait until it receives a command over the communications line, answering the telephone if ringing is detected and hanging up if no valid messages are received within a period of 45 seconds. If an unrecoverable parity error on the tape is encountered during the transmission process, a special message will be sent in place of the record to indicate that the error has occurred. The tape is left after the bad record so the other station may continue reading the tape if so desired. Similar special messages are generated for end of tape and end of file conditions. The SEND program may be commanded to hang up the telephone or rewind the tape and may be requested to return a status message indicating the state of the tape (positioned before the first record or not).

ADDRESS allows the operator to change the address of the communications routine (SEND program). The letters A through Z are valid addresses. Depressing only the ENTER key will cause escape from the ADDRESS function.

12.4 Sample Programs

```
        .DATABUS 3 PROGRAM
        .
        .TESTS DATABUS 6 COMMUNICATIONS WITH DATABUS 3
        .
14400   INQ     INIT 5
14404   REW     INIT 010
14410   HUP     INIT 033
14414   ACK     INIT 6
14420   NAK     INIT 025
14424   EOF     INIT 020
14430   PARITY  INIT 021
14434   EOT     INIT 022
        .
14440   NUMBER  DIM 15
14462   MSG     DIM 100
14631   ADR     INIT "A"
14635   TWO     INIT "2"
14641   ONE     INIT "1"
14645   ALT     INIT "1"
14651   RXALT   INIT "0"
        .
14655           CALL DIAL
14657           GOTO START
14661   REWND   SEND REW,ADR
14665   START   SEND INQ,ADR
14671           RECEIVE 2,MSG
14675           GOTO START IF ERROR
14700           DISPLAY *H1,*V12,"STATUS: ",MSG
14717           CMATCH MSG, "R"
14722           GOTO REWIND IF EQUAL
14725           CMATCH MSG, "N"
14730           GOTO REWIND IF EQUAL
14733           CMATCH MSG,"L"
14736           GOTO START IF NOT EQUAL
        .
14741   SACK    SEND ACK,ADR
14745           GOTO GETHUP
14747   SNAK    SEND NAK,ADR
14753   GETHUP  DSENSE
14754           GOTO GETMSG IF NOT EQUAL
14757           RECEIVE 2,RXALT,MSG
14764           SEND HUP,ADR
14770           CALL DIAL1
14772   GETMSG  RECEIVE 2,RXALT,MSG
14777           GOTO SNAK IF ERROR
15002           CMATCH RXALT,ALT
15005           GOTO SACK IF NOT EQUAL
15010           CMATCH PARITY TO MSG
15013           GOTO PFAIL IF EQUAL
15016           CMATCH EOF TO MSG
15021           GOTO DONE IF EQUAL
```

```
15024          CMATCH EOT TO MSG
15027          GOTO EOTM IF EQUAL
15032          DISPLAY *H1,*V12,MSG
15041  FLIP    MATCH ALT,ONE
15044          GOTO ALTWO IF EQUAL
15047          MOVE ONE,ALT
15052          GOTO TFW
15054  ALTWO   MOVE TWO,ALT
15057  TFW     KSENSE
15060          GOTO REWIND IF EQUAL
15063          GOTO SACK
15065  PFAIL   DISPLAY *H1,*V12,"*** PARITY ERROR ON TAPE ***"
15127          BEEP
15130          GOTO FLIP
15132  EOTM    DISPLAY *H1,*V12,"*** END OF TAPE ***"
15163  DONE    DISPLAY *H1,*V12,*R,"END OF TRANSACTION"
15214          SEND HUP,ADR
15220          DSCNCT
15221          STOP

15222  DIAL    DSCNCT
15223          BEEP
15224          KEYIN *H1,*V12,"PHONE NUMBER: ",NUMBER
15251  DIAL1   DISPLAY *H1,*V12,*R,"I'M DIALING",*R
15274          DIAL NUMBER
15276          CONNECT
15277          SUB I1,I1
15302  DIAL2   ADD 1 TO I1
15305          SEND INQ,ADR
15311          RECEIVE 2,MSG
15315          RETURN IF NOT ERROR
15317          COMPARE 10 TO I1
15322          GOTO DIAL2 IF LESS
15325          KSENSE
15326          GOTO DIAL IF EQUAL
15331          GOTO DIAL1
15333          STOP

15334  DIAL
15336  START
15340  REWIND
15342  SACK
15344  GETHUP
15346  SNAK
15350  GETMSG
15352  DIAL1
15354  PFAIL
15356  DONE
15360  EOTM
15362  FLIP
15364  ALTWO
15366  TFW
15370  DIAL2
```

DATABUS 6 COMMUNICATIONS PROGRAM

15372	INQ
15374	REW
15376	HUP
15400	ACK
15402	NAK
15404	EOF
15406	PARITY
15410	EOT
15412	NUMBER
15414	MSG
15416	ADR
15420	TWO
15422	ONE
15424	ALT
15426	RXALT