WANG LABORATORIES (CANADA) LTD.
49 Valleybrook Drive
Don Mills, Ontario M3B 2S6
TELEPHONE (416) 449-2175
Telex: 069-66546

WANG EUROPE, S.A.
Buurtweg 13
9412 Ottergem
Belgium
TELEPHONE 053/704514
Telex: 26077

WANG DO BRASIL COMPUTADORES LTDA.
Rua Barao de Lucena No. 32
Botafogo ZC-01 20,000
Rio de Janeiro RJ
Brasil
TELEPHONE 226-4326
Telex: 2123296

WANG COMPUTERS (SO. AFRICA) PTY. LTD.
Corner of Allen Rd. & Garden St.
Bordeaux, Transvaal
Republic of South Africa
TELEPHONE (011) 48-6123

WANG INTERNATIONAL TRADE, INC.
836 North Street
Tewksbury, Massachusetts 01876
TELEPHONE (617) 851-4111
TWX 710-343-6769
Telex: 94-7421

WANG SKANDINAVISKA AB
Pyramidvaegen 9A
S-171 36 Solna,
Sweden
TELEPHONE 08-826814
Telex: 11498

WANG NEDERLAND B.V.
Damstraat 2
Utrecht, Netherlands
(030) 93-09-47
Telex: 47579

WANG PACIFIC LTD.
902-3 Wong House
26-30, Des Voeux Road, West
Hong Kong
TELEPHONE 5-435229
Telex: 74879

WANG INDUSTRIAL CO., LTD.
110-118 Kuang-Fu N. Road
Taipei, Taiwan
TELEPHONE 784181-3
Telex: 21713

WANG GESELLSCHAFT M.B.H.
Formanekgasse 12-14
A-1190 Vienna, Austria
TELEPHONE 36.32.80
Telex: 74640

WANG S.A./A.G.
Markusstrasse 20
CH-8042 Zurich 6
Switzerland
TELEPHONE 41.1.26.6866
Telex: 59151

WANG COMPUTER PTY. LTD.
55 Herbert Street
St. Leonards, 2065
Australia
TELEPHONE 439-3511
Telex: 25469

WANG ELECTRONICS LTD.
1 Olympic Way, 4th Floor
Wembley Park,
Middlesex, England
TELEPHONE 01/903/6755
Telex: 923498

WANG FRANCE S.A.R.L.
Tour Gallieni, 1
78/80 Ave. Gallieni
93170 Bagnolet, France
TELEPHONE 33.1.3602211
Telex: 68958F

WANG LABORATORIES GMBH
Moselstrasse 4
6000 Frankfurt AM Main
West Germany
TELEPHONE (0611) 252061
Telex: 04-16246

WANG COMPUTER SERVICES
836 North Street
Tewksbury, Massachusetts 01876
TELEPHONE (617) 851-4111
TWX 710-343-6769
Telex: 94-7421

24 Mill Street
Arlington, Massachusetts 02174
TELEPHONE (617) 648-8550

WANG LABORATORIES, INC.
836 NORTH STREET, TEWKSBURY, MASSACHUSETTS 01876.
TEL. (617) 851-4111. TWX 710 343-6769. TELEX 94-7421

Printed in U.S.A.
700-3030C
10-75-10M
Price $2.50

SYSTEM 2200

BASIC LANGUAGE

POCKET GUIDE

WANG

SYSTEM 2200

BASIC LANGUAGE POCKET GUIDE

WANG LABORATORIES, INC.

836 NORTH STREET, TEWKSBURY, MASSACHUSETTS 01876,
TEL. (617) 851-4111, TWX 710 343-6769, TELEX 94-7421

HOW TO USE THIS POCKET GUIDE

This pocket guide has been written as a companion to the Wang Hardware Pocket Guide. The material in this guide is restricted to the salient features of BASIC available on Wang systems. For more information on BASIC, refer to the Wang BASIC Reference Manual; for elementary descriptions of BASIC and its use for programming, see the BASIC Programming Manual.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

INTRODUCTION

The BASIC language which has been developed at Wang Laboratories, Inc. as the mechanism for supporting its System 2200 hardware has many unique features. Starting as an outgrowth of the computer language developed at Dartmouth College, Wang BASIC has become a sophisticated and powerful tool for the knowledgeable programmer. The fundamentals of BASIC can be learned so rapidly that even novice programmers soon become experts.

NO 'MICKEY-MOUSE' SYMBOLS

Wang BASIC uses English-language words throughout; there are no 'mickey-mouse' symbols to learn and use. BASIC statements are therefore easy to remember and coding is simplified. When it is necessary to edit or debug Wang BASIC programs, the Edit features and Wang BASIC itself are tools that are easy to learn and manipulate. Part of the BASIC substructure contains close to one hundred user-oriented error-messages or diagnostics which are automatically displayed on the video display CRT whenever a programming or processing error occurs.

NUMERICS TO 13 DIGITS; MAGNITUDE 10^{-99} TO 10^{99}.

Numeric values up to 13 digits long can be stored to full precision in the system. A value potentially larger (for example, as the result of a computation) is converted internally to exponential (floating point) format. In this format, values between 10^{-99} and 10^{99} can be stored, with mantissas to 13 digits.

VARIABLES AND FUNCTIONS BOTH NUMERIC AND ALPHANUMERIC

Wang BASIC provides the facility to use numeric and alphanumeric variables; input and output data can contain values and words. Printed output can contain report titles names, etc., directly programmable in a single output statement. The usual math functions are part of the system and operate on

numeric variables (SIN, COS, LOG, exponentiation, etc.). Alphanumeric functions to operate on alpha variables are also available as part of the system. For example, STR, LEN are alpha functions which can examine a substring of the values stored in alpha variables (STR), or count the number of characters in the variable (LEN).

AUTOMATIC FORMATTING OF RECORDED OUTPUT

Whether recorded output is to be stored on tape cassette or disk, the system automatically formats the output recording for the user. The user need only specify (in a SAVE or DATASAVE statement, for example) *what* is to be stored; the system automatically organizes the material in the appropriate fashion, and records the data with control codes to specify what it is. When it is time to extract the recorded information and store it again in memory, the system automatically performs the reverse operation (for example, when a LOAD or DATALOAD statement is executed).

OPTIONAL AUTOMATIC FORMATTING OF PRINTED OUTPUT

The PRINT statement automatically formats any printed output, whether numeric or alphanumeric. when explicit columnar formatting is needed, the PRINTUSING statement can be used.

USER CONTROL OF STORAGE FOR VARIABLES

To conserve memory, the user can restrict the storage space allocated for a particular variable. For example, when an alphanumeric variable or an element of an alpha array is used, the system allocates 16 bytes of storage for it. If the user knows that it will never be larger than 5 bytes, say, the variable can be so defined, by using a DIM or COM statement as follows: A$5 or A$(10)5, depending on whether the

variable is a scalar or an array. Numeric values (normally requiring 8 bytes of storage) can be packed into alpha variables using less than 8 bytes/value if less than 13 digits of mantissa are needed. PACK or $PACK does the job. (PACK, not available on a 2200A, is available on a 2200S or WCS/10 with the Advanced Programming Statements. $PACK is available as one of the General I/O Instruction Set and in a WCS/20 or WCS/30.)

PROGRAMS RECORDED CAN BE NAMED AND PROTECTED

English names which are easy to use and remember can be used to distinguish programs recorded on disk or tape cassette. The name is automatically recorded in a label on the output medium and automatically interrogated when later recalled.

Programs can be protected from accidental overwriting by mechanical protect tabs on tape cassette or diskette.

VARIABLES CAN BE INTERROGATED DURING PROGRAM EXECUTION

Immediate Mode can be used to perform computations or PRINT any program variable when a program has been temporarily halted. The variable then contains some current value, not always its final value. This is an invaluable and convenient debugging tool.

BIT AND BYTE MANIPULATIONS

A variety of manipulations can be performed on variables and individual bytes with a group of statements available on the more advanced systems (not on 2200A; on a 2200S, WCS/10 only with Advanced Programming Statements). With these statements, logical operations (AND, OR, XOR, BOOL) and binary addition can be performed (ADD), a variable or all elements of an array can be set to any binary-bit configuration (INIT), the number of bytes in an alpha variable that represent a numeric value can be

counted (NUM), a numeric value can be placed in an alpha variable (and vice versa) (PACK and UNPACK), the position of a byte that satisfies a specific relation can be found (POS), the individual bits of a byte can be rotated (ROTATE), numeric variables can be changed to alphanumeric (and vice versa) (CONVERT), a character byte can be converted to a floating point number (VAL), and the integer value of an expression can be converted to a character byte (BIN).

These many statements have innumerable applications in more advanced programming schemes and greatly facilitate data manipulations.

POWERFUL SINGLE STATEMENTS PERFORM MANY OPERATIONS

Operations which require whole programs or routines on most systems are executed with certain single statements.

A random number generator is called with the RND statement; it puts out a random number between 0 and 1 whenever it is called.

User functions can be defined with the DEFFN statement. Once defined, such a function, no matter how complex, can be used repeatedly in the program with a single FN call.

In the more advanced systems (not on 2200A, B, C, S or WCS/10 without certain options) a number of additional statements are available. They include the following features.

Matrix inversion in a single statement (MAT INV) eliminates the need for an entire routine to perform this complex computation.

Input for all elements of an array (MAT INPUT) eliminates the need for nested FOR/NEXT loops.

Matrix transposition in a single statement (MAT TRN) also eliminates all need for loops to perform this operation.

Microcoded array search, copy and sort statements (MAT SEARCH, MAT COPY, MAT SORT) perform exceptionally rapid array manipulations and can be used for a variety of applications. In each case, a single statement activates the operation or operations for an entire array, no matter how large.

Programmable I/O control for non-standard devices can be effected with $GIO. This makes it possible not only for a large variety of devices to be connected to a Wang CPU, but also for such devices to be operated under program control. Rapid code translation is accomplished with the $TRAN statement.

IMMEDIATE MODE

In the IMMEDIATE MODE, a Wang System can immediately execute one-line calculator problems. A user can enter a line of statements without a statement number and the line is executed immediately. In the IMMEDIATE MODE a user can also examine the values of variables during program execution or when it has been halted.

EXAMPLES OF IMMEDIATE MODE CALCULATIONS

(1) Print the results of 17.3 raised to the 1.6 power.

```
PRINT 17.3 ↑ 1.6   EXEC
  95.691588717
```

(2) Display the current values of the variables A, B, and C.

```
PRINT A; B; C
  24.921  37.2  95.61
```

MULTI-STATEMENT LINES

The ability to place several statements on a single line makes the IMMEDIATE MODE a very powerful calculating device. Statements are separated by colons.

```
FOR I=1 TO 10:PRINT I,LOG(I) :NEXT I   EXEC
```

Ten values of I and LOG I are printed immediately.

PROGRAMMING MODE

LINE NUMBERS - a line number must be assigned to each program line as it is entered. When the program is run, the System executes the lines in numerical order. When writing a program, it is not necessary to enter lines in numeric sequence.

BRANCHING - Unconditional branching is done with the GOTO statement. Conditional branching is done with the IF statement.

Subroutine transfers can be made in two ways:

(1) GOSUB XXXX

Where XXXX is any valid line number representing the first statement in the subroutine.

(2) GOSUB'XX (expression, expression, . . .)

Note: XX = 00 to 255

The transfer is made to the corresponding DEFFN'XX statement; if arguments are specified, they are transferred to the subroutine.

or

A subroutine with an entry point defined by a special function key can be executed by depressing one of the 32 special function keys after keying in values for the optional arguments.

DEBUGGING AND EDITING

The System contains powerful editing and debugging features:

(1) The CRT is capable of displaying up to 15 lines of program statements at a time (see LIST, LIST S).

(2) Programs or sections of programs can be renumbered using a single RENUMBER statement.

```
:170 X = 10
:171 S = X + SIN (30)
:172 IF S < 50 THEN 170
:RENUMBER 170, 200
:LIST
:200 X = 10
:210 S=X+SIN(30)
:220 IF S < 50 THEN 200
```

(3) An entire line is deleted by entering the line number followed by EXEC

```
:50 PRINT "ABCD"
:50 EXEC                    (deletes line 50)
```

(4) A line can be replaced by entering its line number with different contents.

```
:50 PRINT "ABC"
:50 PRINT "ABCD"
```

(5) A line can be inserted by entering a new line number between two existing line numbers.

```
:100  X=10
:110  Z=X+Y
:105  Y=15
```

(6) A programmable TRACE displays all program transfers and all variables that receive new values when a program is run (see TRACE, TRACE OFF).

(7) An error pointer displays the location of an error. Error 05 illustrates missing right parentheses.

```
10 PRINT SIN(21
             ↑ERR 05
```

(8) HALT/STEP – The HALT/STEP key stops program execution after completion of the current statement. Variables can be examined by using PRINT statements in the Immediate Mode. The program can be executed one statement at a time by depressing the HALT/STEP key; each statement is executed and displayed. Normal program execution can be continued by depressing the CONTINUE and EXEC keys.

(9) The current line being entered is deleted by pressing the LINE ERASE key.

(10) CHARACTER DELETION – Single keystroke entries can be deleted by touching the BACK-SPACE (←) key if the line has not yet been entered into memory.

(11) EDIT MODE – Edit Mode can be entered by pressing the Edit Key; an asterisk (*) replaces the usual colon at the beginning of the line. Once in Edit Mode, the usual Special Function Keys can be used to recall a line from memory, to insert or delete characters, to erase the end of a line and to move the cursor along the current line. Four keys are provided to move the cursor in one-character (←, →) or five character (----→, or ←----) increments. This eliminates rekeying an entire line when changes must be made. Once a line has been corrected and entered, the system automatically drops out of Edit and the Special Function Keys revert to their former use.

Edit Mode is obtained on a 2200A or B as an option (OP-3) and is standard on all other systems.

NON-PROGRAMMABLE COMMANDS

BASIC commands provide the user with a means of controlling the system effectively. Commands are used primarily to instruct the system to perform actions on BASIC programs, such as listing, loading, renumbering.

BASIC commands are entered one line at a time. They differ from BASIC statements becuase they are not preceded by line numbers, and only one command can be entered on each line. All the BASIC commands are executed immediately and not stored in memory.

CLEAR

Removes all program text and or variables from memory.

CLEAR V – *Removes all variables (both common and noncommon) from memory.*

CLEAR N – *Removes all non-common variables from memory; names, attributes, and values of common variables are not changed.*

CLEAR P 120, 150 – *Removes specified program text from memory.*

CONTINUE

Continues program execution after execution has been halted by a STOP verb or use of the HALT/STEP key.

HALT/STEP

Halts program execution after the completion of the current statement. Subsequently, each time HALT/STEP is touched, the next program statement is displayed and executed.

LIST

Prints or displays program text in line number sequence.

LIST S – *Lists program text in increments of fifteen lines on the CRT.*
(Depress EXEC for each successive 15 lines.)

LIST 30,50 — *Lists the current program starting at line 30 and ending at line 50.*

LIST 50 — *Lists program line 50.*

RENUMBER

Renumbers all or a portion of a program.

RENUMBER 100, 150, 5 — *Changes statement number 100 to 150, and renumbers remaining program lines in increments of 5.*

RESET KEY OR BUTTON

Immediately stops program listing or execution, clears the CRT and returns control to the user. The program text is not lost, and all program variables are maintained with the current values.

RUN — *Starts program execution; all non-common variables are initialized.*

RUN 30 — *Starts program execution at the specified statement; variables are not re-initialized.*

SPECIAL FUNCTION KEYS

The Special Function Keys can be defined as entry points for user-defined subroutines, or as character strings for program text entry. Depressing a specified function key causes the corresponding DEFFN' subroutine or DEFFN' text entry statement to be executed.

STATEMENT NUMBER KEY

Sets the number of the line about to be entered equal to the highest current line number + 10.

NUMERICS AND ALPHANUMERICS

ARITHMETIC EXPRESSIONS

An expression is an algebraic formula consisting of any valid combination of numbers, numeric scalar or numeric array variables, and/or numeric functions and operators.

Examples:

```
16.5
4*A + SIN (30)
J
FN2 (X)/(B ↑ 2 – 3)
LOG (9)
I*(J–TAN (2.8))
```

ARITHMETIC OPERATORS

Symbol	Sample Formula	Explanation
↑	A ↑ B	Raise A to the power of B
*	A * B	Multiply B by A
/	A / B	Divide A by B
+	A + B	Add B to A
–	A – B	Subtract B from A

RELATIONAL SYMBOLS

Symbol	Sample Relation	Explanation
=	A = B	A is equal to B
<	A < B	A is less than B
<=	A <= B	A is less than or equal to B
>	A > B	A is greater than B
>=	A >= B	A is greater than or equal to B
<>	A <> B	A is not equal to B

ASSIGNMENT SYMBOL

Symbol	Sample Relation	Explanation
=	A = 2.5	Assign the value 2.5 to the variable A

ALPHANUMERIC LITERALS

Alphanumeric literals are character strings enclosed by double quotation marks. They are used in PRINT statements to provide alphanumeric output for labeling purposes and in LET and IF statements to set or compare alpha variable values. Literals enclosed in single quotes also are permissible to provide lowercase alphanumeric output.

```
100 PRINT "ANSWER=";A
510 PRINT "M"; 'R'; "S"; 'MITH'
```

HEXADECIMAL FUNCTION (HEX)

The HEX function is a form of literal that enables use of any 8-bit codes in BASIC programs. Each character is indicated by two hexadecimal digits (0-9, A-F). The HEX function can be used wherever literals enclosed in double quotes can be used.

```
100 PRINT HEX(0C)
200 IF A$ > HEX(3132) THEN 50
```

ALPHANUMERIC FUNCTIONS

Four functions are available to operate on alpha scalars and arrays:

STRING FUNCTION (STR)

Wang BASIC provides a function which permits the user to extract, examine, compare or replace a specified portion of an alphanumeric variable. The STR function operates on alphanumeric variables, and can be used in any BASIC statement where alphanumeric variables are permissible.

Assuming B$ = "ABCDEFGH", the statement

10 A$ = STR(B$,2,4)	sets A$ equal to the substring "BCDE"
20 STR(C$, 1,4) = B$	The first four bytes of C$ receive the first four characters of B$.

LENGTH FUNCTION (LEN)

This function is used to determine the length of the specified alphanumeric variable; that is, the number of characters it contains excluding trailing spaces.

```
210 X = LEN(A$)
```

NUM (not on 2200A, on WCS/10 or 2200S only with Advanced Programming Statements)

Determines the number of sequential ASCII characters in the alpha variable that represent a valid numeric value. Includes trailing spaces in the count.

```
20 X = NUM (A$)
40 X = NUM (STR (A$, 7, 3))
```

POS (not on 2200A, on WCS/10 or 2200s only with Advanced Programming Statements)

Finds the position of the first character in the specified alphanumeric value that satisfies the specified relation.

```
10 X = POS (A = "S")
20 Y = POS (STR(B$, 1, 10) <> 7E)
```

NUMERIC FUNCTIONS

System-defined math functions are:

SIN (expression)	Find the sine* of the expression
COS (expression)	Find the cosine* of the expression
TAN (expression)	Find the tangent* of the expression
ARC SIN (expression)	Find the arcsine* of the expression
ARC COS (expression)	Find the arccosine* of the expression
ARC TAN (expression)	Find the arctangent* of the expression
EXP (expression)	Find e to the power of the expression
LOG (expression)	Find the natural logarithm of the expression
ABS (expression)	Find the absolute value of the expression
SQR (expression)	Find the square root of the expression
RND (expression)	Produce a random number between 0 and 1
INT (expression)	Reduce the expression to an integer by finding greatest integer $\leqslant$ expression; if zero, set equal to zero.
SGN (expression)	Assign the value 1 to any positive number, 0 to zero and −1 to any negative number
#PI	Assign the value π (3.14159265359)

*Arguments of trigonometric functions are normally assumed to be radians. Degrees, radians or gradians can be chosen by executing a SELECT statement.

SELECT R EXEC (select RADIANS as argument)
SELECT D EXEC (select DEGREES as argument)
SELECT G EXEC (select GRADS as argument)

USER-DEFINED FUNCTIONS

These functions allow the programmer to define his own functions in a program.

The function is identified by the letters FN followed by a single letter or digit, and an expression or dummy variable enclosed in parentheses. The function is defined in a DEFFN statement.

```
:100 DEFFNA (X)=X+1
:110 Z=3*FNA(5)↑2+R
```

VARIABLES

Both numeric and alphanumeric variables can be used in Wang Systems. Variables can either be scalars (not subscripted) or arrays (subscripted in one or two dimensions). Certain system-defined functions operate on numeric variables (SIN, COS, etc.), while others operate on alpha variables (STR, LEN, etc.).

NUMERIC SCALAR VARIABLE NAMES

Scalar (nonsubscripted) variables: single letter optionally followed by a single digit; e.g., A or A1 but not AA or 1A.

ALPHANUMERIC SCALAR VARIABLE NAMES

Same specifications as those for numeric scalar variable names except the name is followed by a dollar sign ($); e.g., A$,A1$.

ARRAYS

Both one- and two-dimensional numeric and alphanumeric arrays can be referenced in Wang BASIC. All arrays must appear in dimension (DIM) or common (COM) statements prior to being referenced. The maximum value for any array dimension is 255; array subscripts can be any valid BASIC expression with a truncated value greater than or = to 1 and less than 256. An array cannot contain more than 4,095 elements.

```
100 DIM A(100,12), D$(255)
100 COM A$(10),B(4,6)
```

NUMERIC ARRAY VARIABLES

Single letter (optionally followed by a single digit) followed by one or two expressions (separated by commas) enclosed in parentheses. The truncated value of the expressions must be ⩾ 1 and less than 256.

```
A(3,4), F3(X,J), B(3)
```

ALPHANUMERIC ARRAY VARIABLES

Single letter (optionally followed by a single digit) followed by one or two expressions followed by a dollar sign ($); e.g., A$(1), B1$(2,3)

ARRAY NAMES

The name of a numeric array is the letter (and optional digit) which refers to an array defined in a DIM or COM statement; e.g., A, F3.

The name of an alpha array is the letter (and optional digit) and dollar sign which refers to an array defined in a DIM or COM statement; e.g., A$, B2$.

ARRAY DESIGNATORS

Array designators are array names followed by empty parentheses; e.g., A(), A1$(), B$(), A1().

LENGTH OF ALPHA SCALARS AND ARRAY ELEMENTS

The length of each alpha scalar or element of an alpha array is 16 bytes, unless specifically dimensioned otherwise. Dimensions of from 1 to 64 bytes can be specified for any alpha scalar or all the elements of an alpha array by following the scalar or array name in a COM or DIM statement with a length.

```
10 COM A$40, B$(2,4)10, C$64
20 DIM B1$(255)1, C$54, F$(10,2)64
```


BASIC STATEMENTS

COM

Stores information in common memory for use in a subsequent program. Array variables are dimensioned by COM and alphanumeric variables can have their length specified.

```
40 COM Y,A(2,3),B$20
```

COM CLEAR (not on 2200A or B, only on 2200S or WCS/10 with OP-24)

Defines previously common variables as non-common variables or defines previously non-common variables as common variables.

```
10 COM CLEAR        ('un-commons' previously
                     common variables)
110 COM A,B,X,Y(10) ('commons' variables)
220 COM CLEAR X     (redefines X and Y as
                     non-common)
```

CONVERT (not on 2200A)

Converts alpha to numeric variables, or numeric expressions to alpha variables.

```
CONVERT X TO A$, (####)
CONVERT A$ TO N
```

DATA

Provides values to be used by the variables in READ statements.

```
50 DATA 4,3,5,6,"NAME"
```

DEFFN

Defines user functions of a single variable. The function definition can be any formula entered on one program line. The variable in parentheses is a dummy variable which can have the same name as another variable in the program without affecting its value.

```
10 DEFFN Y(X)=2*X↑3.2+5*SIN(X)
```

DEFFN'

Defines the beginning of a subroutine within a program executed either by manually depressing the indicated Special Function Key or under program control by using the GOSUB' statement. Optionally, values can be passed to the subroutine.

```
100 DEFFN' 1 (X,Y)
110 Z=3*(X+3.5)↑2+2*(Y+6.4)↑2
120 RETURN
```

This special function subroutine would be entered by keying the values for X and Y separated by a comma and then depressing Special Function Key 01, or by a program call such as:

```
200 GOSUB' 01 (P,R)
```

The DEFFN' statement also can be used to define the Special Function Keys for customized text entry.

```
10 DEFFN' 2 "REWIND"
```

When Special Function Key 02 is depressed, REWIND appears on the CRT as if it were keyed in.

Any 8-bit code can be entered via special function keys using the HEX function in a DEFFN' statement.

```
100 DEFFN' 3 HEX(5F)
200 DEFFN' 5 HEX(2C4B2F)
```

DIM (dimension)

Reserves space for one- or two-dimensional arrays. The DIM statement also can be used to set the maximum length of alphanumeric variables.

```
50 DIM A$32, B$(5,2) 24
20 DIM I (45),J(10,10)
```

END

END is an optional statement. It displays the amount of free space left in memory.

```
900 END
```


FOR

Specifies the beginning of a loop.

```
200 FOR I = 1 TO 10
210 FOR R = 2 TO 10 STEP 2
220 FOR X = 100 TO 500 STEP 10
    .
    .
    .
240 NEXT X
250 NEXT R
260 NEXT I
```

GOSUB

Specifies a transfer to the first line of a subroutine.

```
10 GOSUB 30
```

GOSUB'

Transfers execution to the subroutine beginning with a corresponding DEFFN' statement. Arguments can be passed with a GOSUB' statement.

```
100 GOSUB' 07 (A,B$)
    .
    .
    .
500 DEFFN' 07 (I,J$)
```

GOTO

Transfers program execution to the specified program line number.

```
50 GOTO 10
```

If GOTO is entered as an Immediate Mode statement, it transfers execution to the specified line and halts before that line is executed. The program then can be executed from that point by touching HALT/STEP, or CONTINUE.

HEXPRINT (not on 2200A)

Prints the value of the alpha variable or of the elements of the alpha array in hexadecimal notation.

```
10 HEXPRINT A$
20 HEXPRINT B$( )
```

IF END THEN

Tests to see if the end of the current data file has been encountered and transfers to the indicated line number if it has been encountered (for use with tape cassette or disk data files).

```
110 IF END THEN 130
```

IF . . . THEN

If the specified relation is true, execution is transferred to the indicated line number. If the relation is false, execution passes to the next sequential statement.

```
40 IF A < B THEN 35
50 IF A > C THEN 100
```

% (Image)

This statement is used in conjunction with a PRINTUSING statement to provide an image line for formatted output. The IMAGE statement contains text to be printed and format specifications used to format parameters contained in the PRINTUSING statement. The image is reused until all PRINTUSING arguments are processed.

```
% AMOUNT: $##,###.##
```

INPUT

Allows the user to supply data via the keyboard during the running of a program already stored in memory.

```
40 INPUT "VALUE OF A,B",A,B
```

KEYIN (not on 2200A)

This statement checks if there is a character ready to come in from the input device buffer and, if one is ready, it reads the character and stores it in the alpha variable. It permits customized input routines to be written.

```
10 KEYIN A$,100,200
```

LET

Assigns values to the variable or variables specified. This verb can be omitted.


```
110 LET J=3
10 X,Y,Z=5.3
50 N$(1,3) = "CLASS #10"
```

NEXT

The NEXT statement signifies the end of a loop begun by a FOR statement.

```
30 FOR X=N TO SQR (N↑3)
    .
    .
    .
100 NEXT X
```

ON (not on 2200A)

The ON statement is a conditional GOTO or GOSUB statement.

```
10 ON I GOTO 10,15,100,900
If I=1 GOTO 10, If I=2 GOTO 15, etc.
200 ON J-3 GOSUB 30,15,73,106
If J=4 GOSUB 30,
If J=5 GOSUB 15, etc.
```

ON ERROR . . . GOTO (not on 2200A or B; only on 2200S, WCS/10 with OP-24)

Bypasses usual system error messages and branches to a specified statement number. Error code and erring statement number are saved.

```
800 ON ERROR E$, N$ GO TO 900
900 REM ERROR RECOVERY ROUTINE
```

PRINT

Prints the values of specified numbers, variables, and expressions in zoned or packed format. Comma separators specify zoned format; semi-colons, packed format.

```
150 PRINT 13E-4, B, 3+A*X
175 PRINT J;K(1,2);K(2,2)
```

TAB (PRINT Function)

Position of printed output information can be controlled by using the TAB() parameter. Cursor or output head is positioned at specified column.

```
300 PRINT X;TAB(X + 20);"*"
```

PRINTUSING

PRINTUSING operates in conjunction with a referenced Image statement. Parameters in the PRINTUSING statement are formatted in the print line as directed by the Image statement.

```
100 PRINTUSING 101,A$,B
101 % ITEM NO. #### COSTS $ ##,###.##
```

READ

The READ statement assigns the values contained in DATA statements to READ statement variables, in sequential order.

```
100 READ A,L$,F(3,1)
```

REM

The REM statement is used to insert comments or explanatory remarks in a program.

```
200 REM PRICING SUBROUTINE
201 REM ANY DATA
```

RESTORE

The RESTORE statement allows the repeated use of DATA statement values by READ statements.

RESTORE n sets a pointer to the nth data value. A subsequent READ statement reads data beginning with this value.

```
100 RESTORE          (start with first value)
100 RESTORE 11       (start with eleventh value)
```

RETURN

Returns processing to the statement following the last executed GOSUB statement.

```
270 RETURN
```

RETURN CLEAR (not on 2200A or B)

Clears return address information generated by last subroutine call executed. A dummy RETURN statement; it eliminates chance of ERR 02, and is particularly useful to avoid returning to keyboard input when entering a program with a Special Function Key.

```
200 RETURN CLEAR
```

SELECT

The SELECT statement is used to select I/O devices for specified operations; to select degree, radian, or gradian arguments for trigonometric functions; and to insert timed pauses into the execution of a program whenever a CR/LF is output.

```
90 SELECT PRINT 215 (80)
Assign device address 215 for PRINT, PRINTUSING or HEXPRINT with line length of 80.
100 SELECT CO 005, CI 001
Assign device address 005 for Console Output, 001 for Console Input.
120 SELECT D
Select degrees as the argument for trig functions.
130 SELECT P1
Select 1/6 sec pause after each line for Console Output, Print.
```

STOP

Causes program execution to terminate. A message can be displayed when STOP is executed.

```
1050 STOP "MOUNT SCRATCH TAPE"
```

TRACE

Provides for tracing of all or a portion of a BASIC program.

```
100 TRACE
```

TRACE OFF

Turns off TRACE mode.

```
150 TRACE OFF
```

TRACE or TRACE OFF can be used either as Immediate Mode statements or program statements. This allows all or a segment of a program to be traced, either during execution or when stepping through it. Trace output appears on the CRT and also can be printed with the device currently selected for Console Output operations.

DATA MANIPULATION

Not on 2200A; available on 2200S or WCS/10 only with Advanced Programming Statements.

ADD

Immediate

Add, in binary, the value specified by two hexadecimal digits to every character in the specified alphanumeric variable.

```
100 ADD (A$,FF)
```

String-to-String

Add, in binary, on a sequential character by character basis, the characters specified in the second alphanumeric variable to the characters specified by the first. Addition takes place right adjusted. High order characters of short strings are assumed to be zero.

```
200 ADD (A$(1),B$)
250 ADD (STR(A$,2,3),C$)
```

ADD C

Immediate

Add, in binary, the value specified by the two hexadecimal digits to the value of the specified alphanumeric variable. (Carry is propagated between characters.)

```
100 ADD C (A$, 3A)
```

String-to-String

Add, in binary, the entire value contained in the second alphanumeric variable to that of the first. Carry is propagated between characters. The addition is right adjusted. The high order portion of a shorter variable is assumed to be zero.

```
10 ADD C (A$,B$)
20 ADD C (C$(1), STR(D$,5,4))
```

AND, OR, XOR

Immediate

A binary logical AND, OR or exclusive OR is formed with the value specified by the two hex digits and each character in the alphanumeric variable; the result is stored in the variable.

```
10 AND (A$, 7A)
20 OR (B$(1), 3E)
30 XOR (STR(C$,4,6),80)
```

String-to-String

A binary logical AND, OR or exclusive OR is formed sequentially character by character with the characters of the first and second alphanumeric variables starting with the first characters in each variable. The result is stored in the first variable.

```
10 AND (A$,B$)
20 OR (A$,B$(3,4))
30 XOR (STR(C$,2,4),D$)
```

BIN

This statement converts the integer value of the expression following the = sign, to a 1-byte binary number and sets the first character of the specified alphanumeric variable equal to that value. BIN is the inverse of the function VAL. The maximum value is 255 (HEX FF).

```
10 BIN (A$) = 64 - 2*X
```

BOOL

Immediate

Performs any of sixteen specifiable logical Boolean operations with the two specified hexadecimal digits and each character in the specified alphanumeric variables; stores the result in the variable.

```
10 BOOL 1 (C$,F0)
20 BOOL 7 (B$,3A)
```

String-to-String

Performs any of sixteen specifiable logical Boolean operations between the characters of the first and second alphanumeric variables on a character by character basis starting with the first character of each variable. Results are stored in the first variable.

```
10 BOOL E (A$,B$)
20 BOOL 6 (STR(A$,4,8),B$)
```

INIT

Sets each character of the specified alphanumeric variable or array to the value indicated by two hexadecimal digits, a character in quotes, or the first character in a specified variable or array.

```
100 INIT(00) A$,B$( ), C$
110 INIT ("B") F$
120 INIT (B$)F$( )
```

NUM Function

Determines the number of sequential ASCII characters in the alpha variable that represent a valid numeric variable. Includes trailing spaces in the count.

```
20 X = NUM (A$)
40 X = NUM (STR (A$,7,3))
```

PACK

Packs numeric variables or arrays into alphanumeric variables or arrays, reducing the storage requirement for numeric data where only a few significant digits are required.

```
10 PACK (+##.####)A$ FROM B
20 PACK (####) A$ FROM X( )
```

POS Function

Finds the position of the first character in the specified alphanumeric value that satisfies the specified relation.

```
10 X = POS (A$ = "$")
20 Y = POS (STR(B$,1,10) <> 7E)
```

ROTATE

Rotates the bits of each character in the specified alphanumeric variable to the left from one to seven places; the high order bits replace the low order bits.

```
10 ROTATE (A$,4)
```

UNPACK

This statement is used to unpack numeric data that was packed by a PACK statement.

```
10 UNPACK (####) A$ TO X,Y,Z
20 UNPACK (+##.##) A$( ) TO X( )
```


VAL Function

Converts the binary value of the first character of the alpha variable to a floating point number.

```
10 X = VAL (A$)
20 Y = VAL (STR (B$,N,1))
```

MATRIX STATEMENTS

Fourteen matrix statements are available to the user in several system configurations. The statements are standard features on any 2200T, WCS/20 or WCS/30; can be obtained as Option 1 on a 2200B or C; and can be obtained as Option 21 on any 2200S or WCS/10. Option 21 is included in Options 22, 23 and 24.

These matrix statements permit the user certain input, output and computational operations on entire arrays with single program statements. Computations are performed according to the rules of linear algebra.

The fourteen matrix statements are:

MAT + (addition)

Adds two numeric arrays of the same dimensions.

```
100 MAT A = A + D
```

MAT CON (constant)

Sets all elements of a numeric array to one (1); the constant matrix. The matrix can be redimensioned.

```
100 MAT A = CON
200 MAT B = CON (5,7)
```

MAT = (equivalence)

Replaces each element of one numeric array with the corresponding element of another numeric array.

```
100 MAT A = B
```

MAT IDN (identity)

Sets a numeric array to the identity matrix (diagonal elements = 1, all other elements = 0); the array can be redimensioned.

```
100 MAT A = IDN
200 MAT B = IDN(5,5)
```

MAT INPUT

Permits the user to input elements for an entire array without the use of FOR/NEXT loops. Arrays can be numeric or alphanumeric and can be redimensioned.


```
100 MAT INPUT A, B(2), C(2,4)
200 MAT INPUT A$(4)3, C$
```

MAT INV, d (inverse)

Inverts a numeric matrix using the Gauss-Jordan Elimination Method. The determinant can be optionally obtained (d).

```
100 MAT B = INV(A), D
200 MAT C = INV(B)
```

MAT * (multiplication)

Stores the product of two numeric arrays.

```
100 MAT A = E * F
```

MAT PRINT

Outputs arrays row by row in zoned or packed format. Arrays can be numeric or alphanumeric.

```
100 MAT PRINT Z$;
200 MAT PRINT A, B$
```

MAT READ

Permits values from DATA statements to be placed in array elements automatically without using FOR/NEXT loops. Numeric or alpha arrays can be used, and the arrays can be redimensioned.

```
100 MAT READ A, B(2,3)
200 MAT READ C(2), D$(4)6
```

MAT REDIM (redimension)

Redimensions the arrays specified.

```
100 MAT REDIMA(4,5)
200 MAT REDIMB$(14)3
```

MAT ( ) * (scalar multiplication)

Stores the product of an expression and each element of an array in another numeric array.

```
100 MAT C = (SIN(X))*A
200 MAT B = (5)*A
```

MAT - (subtraction)

Subtracts numeric matrices of the same dimension.

```
100 MAT C = A - B
```

MAT TRN (transpose)

Transposes a numeric array; the array can be redimensioned automatically.

```
100 MAT C = TRN(A)
```

MAT ZER

Sets all elements of a numeric array to zero; the matrix can be redimensioned.

```
100 MAT C = ZER (5,2)
```


GENERAL I/O STATEMENTS

General I/O Statements are standard features of any 2200T, WCS/20, or WCS/30 and can be obtained as an option on certain other configurations; as Option 2 on a 2200B or C; as Option 23 on a 2200S or WCS/10 and as part of Option 24.

These statements provide certain I/O control functions particularly useful for the programmable operation of non-Wang peripherals. The statements are:

$GIO

This is a generalized I/O statement designed to perform data input and output as well as I/O control operations with a programmable signal sequence.

```
100 $GIO (6CFA 4400 A206 8607,B$) A$( )
```

$IF ON

This statement tests the ready/busy condition of an I/O device and can execute a conditional branch to a specified line number.

```
$IF ON #3,200
```

$TRAN

This statement facilitates high-speed character-code translations or specific character replacement via a table stored in an alphanumeric variable or array.

```
100 $TRAN (A$, B$) 2F
```

$PACK and $UNPACK

These are complementary statements that either pack numeric or alphanumeric values into a record or unpack such values from a record. A variety of formats can be used and the operation can be defined by field width or delimiters.

```
100 $PACK (D=X$) B$( ) FROM A,B,C
200 $UNPACK (F=A$) B$( ) TO X,Y,Z,D$
```

SORT STATEMENTS

Six Sort Statements can be obtained in several configurations to provide rapid array search, copy and sort capabilities. The statements are:

MAT CONVERT

Converts numeric to alpha data and puts values in sort format, pre-processed to optimize sort operations.

```
MAT CONVERT N( ) TO A$( ) (1,5)
```

MAT COPY

Transfers data byte-by-byte from a portion of an alpha array to another alpha array.

```
MAT COPY – A$( ) TO B$( ) < 5, 10>
```

MAT MERGE

Performs a segment of a sort/merge operation by combining several pre-sorted arrays.

```
MAT MERGE A$( ) TO W1$( ), W2$( ), L$( )
```

MAT MOVE

Moves data from one array to another according to the subscripts in the locator array. The locator array must be generated by a MAT SORT or MAT MERGE statement and a specific field within each element of an alpha array can be moved.

```
MAT MOVE A$( ) (3,2), L$(1) TO A2$(1)
```

MAT SEARCH

Scans the input array for substrings of characters that satisfy the relation defined and stores the locations of each such substring in the locator array.

```
MAT SEARCH A$( ),<B$ TO L$( ) STEP 2
MAT SEARCH A$( ) <S,N>, > STR (Q$,3,5) TO S$( )
```

MAT SORT

Takes the elements of the input array and creates a locator array of subscripts according to ascending order.

```
MAT SORT G$( ) TO W$( ), G1$( )
```


TAPE CASSETTE DRIVES

BASIC logic to support tape cassette drives is available on all systems.

BACKSPACE

Backspaces the specified tape cassette any number of logical records or complete files or backspaces to the beginning of the current data file.

```
500 BACKSPACE #1,3
100 BACKSPACE/10B, 4F
550 BACKSPACE BEG
```

DATALOAD

Reads one logical record from the designated tape and assigns the data values read to the variables and/or arrays in the argument list. Data files also can be searched out by name.

```
10  DATALOAD #2,A( ), B, Z$( )
100 DATALOAD "NAME 2"
200 DATALOAD/10C, A, B, C$, E( )
```

DATALOAD BT (not on 2200A, not on 2200S or WCS/10)*

This statement reads the next block of 100 or 256 bytes from cassette tape and stores the information in the specified alphanumeric array, ignoring recording control information.

```
100 DATALOAD BT A$( )
200 DATALOAD BT (N=100) C$( )
```

DATARESAVE

The DATARESAVE statement allows the user to update (rewrite) records, in place; both data and the header records of an existing data file can be rewritten.

```
100 DATARESAVE OPEN "NAME 2"
200 DATARESAVE B( ), X$
300 DATARESAVE/10B, A, B, C$, D$
```

*can be obtained on WCS/10 or 2200S with OP-22, 23 or 24.

DATASAVE

The DATASAVE statement causes the values of variables, expressions and array elements to be recorded sequentially onto the specified tape as a logical record. Data files can be opened, named (by writing a header record), and closed (by writing a trailer record).

```
10 DATASAVE OPEN "NAME"
10 DATASAVE END
10 DATASAVE #2,N$,B,K( )
10 DATASAVE A,B,C$,E( )
```

DATASAVE BT (not on 2200A, 2200S or WCS/10)*

This statement records a block of data (100 or 256 bytes) on cassette tape with no control information (if N is not specified, 256 is assumed).

```
DATASAVE BT A1$( )
DATASAVE BT (N=100) B$( )
```

LOAD (command)

Loads the program file currently positioned in the tape drive into the System or searches the tape for the named program and loads it.

LOAD – Loads current program file from cassette.

LOAD "NAME" – Searches tape for the designated program file and loads it.

```
LOAD/10B, "NAME"
```

LOAD (statement)

The LOAD statement is used to chain programs or subroutines from cassette tapes.

When executed in a program, LOAD stops program execution, deletes all or part of the current program, clears non-common variables, loads the new program or program segments and starts execution at the new program.

*can be obtained on WCS/10 or 2200S with OP-22, 23 or 24.


```
400 LOAD
500 LOAD "NAME" 60,350
600 LOAD /10B, 200
```

REWIND

Rewinds the cassette in the specified drive.

```
100 REWIND
200 REWIND /10B
300 REWIND #4
```

SKIP

Allows the user to skip any number of program and data files, or logical data records, on the cassette in the specified drive. The user can also skip to the end of the current data file.

```
100 SKIP 3F
500 SKIP 4
350 SKIP END
400 SKIP /10B, 4F
500 SKIP #2,6
```

SAVE (command)

Records an entire program or a specified portion of a program onto tape cassette as a program file.

```
SAVE "INDEX"
```

PUNCHED TAPE READER

(Not on 2200A; available on 2200S or WCS/10 only with Options 22, 23 or 24.

DATALOAD

This statement reads values from punched paper tape and sequentially assigns those values to the variables in the argument list. Numeric information must be ASCII characters in legal format. Values must be separated by CR/LF characters.

```
DATALOAD X, Y, A$, B$
```

DATALOAD BT

This statement reads paper tapes in any format forwards or backwards and stores the characters that are read in the alpha variable or alpha array designator specified. The tape is read until a specified stop character is encountered, or the alpha variable or array is full, or the number of characters specified by N are read, whichever occurs first. All eight channels of the paper tape are read.

```
DATALOAD BT (N=100) A$( )
DATALOAD BT (S=FE) B$
```

LOAD (command)

The LOAD command with the tape reader address specified, causes the program punched on the paper tape to be loaded and appended to the current program in memory. The program must be punched in ASCII characters.

```
LOAD /618
```

LOAD (statement)

The LOAD statement is used to chain programs or subroutines recorded on punched tape.

```
100 LOAD
500 LOAD /618, "NATL" 60,350
```


TELETYPE

(Not supported on 2200A; available on 2200S or WCS/10 only with Options 22, 23 or 24.)

DATALOAD

This statement reads values from the Teletype® paper tape and sequentially assigns those values to the variables in the argument list. Tape must be punched in format produced by DATASAVE statement.

```
DATALOAD X,Y,A$,B$
```

DATALOAD BT

This statement reads a paper tape and stores the characters read in the alpha scalar variable or alpha array specified. This command facilitates the reading of paper tape in any code or format. Reading is terminated when the array is full or a specified character is read (N) or a specified stop code is read (S).

```
DATALOAD BT (N=30) A$( )
DATALOAD BT (N=40, S=FA) #3, B$
```

DATASAVE

This statement causes the values specified in the argument list to be punched on paper tape. Numeric values are written in a form identical to that resulting from a PRINT statement.

```
DATASAVE #3, OPEN "T FILE"
DATASAVE X,Y,A$
```

DATASAVE BT

This statement punches the values of an alpha variable or alpha array onto a paper tape with no control information (i.e., no CR/LF RUBOUT RUBOUT separating values). Trailing spaces in alpha values are punched.

```
DATASAVE BT #2, A$( )
```

® Registered Trademark, Teletype Corporation

LOAD (command)

When the LOAD command is entered, the program punched on the paper tape is loaded and appended to the current program in memory.

```
LOAD /41D
```

LOAD (statement)

The LOAD statement is used to chain programs or subroutines from the Teletype paper tape reader.

```
100 LOAD /41D
140 SELECT TAPE 41D
150 LOAD 70,300
```

SAVE (command)

The SAVE command causes BASIC programs (or portions of BASIC programs) to be punched on paper tape.

```
SAVE /41D,100,200
```


CARD READERS

INPUT

This statement allows the user to supply input data via the Mark Sense or Punched Card Reader during the running of a program already stored in memory.

```
110 SELECT INPUT 517
120 INPUT A,B2 C(3)
```

CONSOLE INPUT

When the Card Reader is selected as the Console Input device, programs and commands can be entered from the card reader as if entered from the keyboard.

```
:SELECT CI 517
```

DATALOAD*

This statement reads values from a card and sequentially assigns those values to the variables in the argument list.

```
DATALOAD A, B$( )
```

DATALOAD BT*

This statement reads 8-bit characters in any code format from a card and stores the characters read in the alpha variable or alpha array designated.

```
100 DATALOAD BT (N=40) A$( )
```

LOAD

This statement loads programs from the card reader.

```
:LOAD /517
```

*Not available on 2200A; supported on 2200S or WCS/10 only with OP-22, 23 or 24.

PLOTTERS

(Not supported on 2200A; available on 2200S or WCS/10 only with Option 22, 23 or 24.)

PLOT

This statement moves the plotting pen or typing element from its current position to a specified position on the plotting surface. Movement can be specified in increments (Δx, Δy). Movement can be made with the pen up or the pen down. When literals or alpha variables are used, they represent labels to be plotted. Multiple plot arguments are legal.

```
10 PLOT <10, 20, HEX(FB)>
20 PLOT <X,Y, "VALUE">, <40, 60, "-">
30 PLOT 10 <X,Y,"-">
40 PLOT <X,Y,U>
50 PLOT <10, 20, D>
60 PLOT < ,, >
```


DISK UNITS

(Not supported on 2200A; available on 2200S or WCS/10 only with Option 24.)

COPY

This statement copies the specified sectors from one platter to the other.

```
COPY RF (100,4070)
```

DATALOAD BA

Loads unformatted data blocks of one sector (256 bytes) from disk into an alphanumeric array.

```
100 DATALOAD BA R(X,Y)C$ ( )
```

DATALOAD DC

Used to read logical data records from a cataloged disk file and to assign the values read to the variables and/or arrays in the argument list sequentially.

```
DATALOAD DC #2, A( ), J$
```

DATALOAD DC OPEN

Used to open data files that have previously been cataloged on disk and to assign the pertinent disk address to the specified file number (i.e., #1, #2, #3 etc.; if not specified, #0 is assumed).

```
DATALOAD DC OPEN F#2, "HEADER"
```

DATASAVE BA

Saves unformatted data on disk in blocks equivalent to one sector (256 bytes) from an alphanumeric array.

```
100 DATASAVE BA F(N,N)A$( )
```

DATASAVE DA

This statement writes a logical data record containing all the data in the argument list onto the disk. The absolute sector addressing mode is used.

```
200 DATASAVE DA R (1000,S)A$,B( )
300 DATASAVE DA F(B,C) END
```

DATASAVE DC

Causes one logical record, comprised of all the data in the argument list to be written onto disk starting at the current sector address associated with the specified file number in the Device Table. The record is written in as many files as is required. It is also used to write an end-of-file record (END).

```
30 DATASAVE DC #2, B$,X( )
40 DATASAVE DC #4, END
```

DATASAVE DC CLOSE

Used to close any single data file or all data files which are currently open and assigned to file numbers (#0, #1, #2, etc.). It in effect, clears this information stored in the file numbers to protect against subsequent erroneous references.

```
304 DATASAVE DC CLOSE #2
500 DATASAVE DC CLOSE ALL
```

DATASAVE DC OPEN

Used to reserve space for new cataloged files in the Catalog Area, to enter the file name and appropriate system information in the Catalog Index, and to assign pertinent disk addresses to the specified file number. It is also used to reserve space for temporary work files outside the Catalog Area or to rename scratch files.

```
100 DATASAVE DC OPEN T #2,300"FIL2"
200 DATASAVE DC OPEN R TEMP, 900, 1000
300 DATASAVE DC OPEN F #4, "TABLE1"
```

DBACKSPACE

Used to backspace over logical records or sectors within a cataloged disk file or backspace to the beginning of the file (BEG).

```
100 DBACKSPACE BEG
```

DSKIP

Used to skip over logical records or sectors in a cataloged disk file or to skip to the present end of file (END).

```
DSKIP #3,4
DSKIP 4*XS
DSKIP END
```


LIMITS

Obtains the beginning and ending sector address and current sector address used for a cataloged file.

LIST DC

This statement displays or prints out a listing of the information contained in the Catalog Index. Listing occurs on the device currently selected for LIST.

```
LIST DC F
```

LOAD DA (command)

Used in loading BASIC programs from disk in absolute sector addressing mode.

```
LOAD DA R (100,S)
LOAD DA T #2, (X,B)
```

LOAD DA (statement)

Used to chain programs or subroutines from disk in absolute sector addressing mode.

```
110 LOAD DA F(X,S)
125 LOAD DA T #3, (4500,S)100,700
```

LOAD DC (command)

For use in loading cataloged BASIC program files from the disk. This causes the system to locate the named program file in the catalog and to append it to the program text currently in memory.

```
LOAD DC F "MAPROGIO"
```

LOAD DC (statement)

This statement is used to chain programs and subroutines from disk.

```
110 LOAD DC F "NXTAM"
 50 LOAD DC T #2 "SUB 1" 50,700
```

MOVE

This statement is used to copy the entire catalog area from one platter to another, deleting all scratched files from the catalog and relocating the active files.

```
MOVE FR
MOVE RF
```

MOVE END

```
MOVE END F = 3000
```

Used to increase or decrease the size of the Catalog Area on a disk.

SAVE DA (command)

Used to save programs or program segments on disk in the absolute sector addressing mode.

```
SAVE DA F(100,S)
SAVE DA R(S,S)300,700
```

SAVE DC (command)

Causes a BASIC program or a portion of a program to be named and recorded on the designated disk. Optionally, additional sectors can be specified for program files.

```
SAVE DC R "DGMENT"
```

SCRATCH

Used to set the status of the named disk files to a scratched condition. The files can be renamed and reused or will be removed when the catalog is transferred with a MOVE command.

```
SCRATCH F "NAM","PASIV"
```

SCRATCH DISK

Reserves space for the Catalog Index and Catalog Area on the disk prior to saving any program files or data files on the disk. This is generally done only once to an initialized disk platter for catalog operations.

```
SCRATCH DISK F LS=2, END 50
```

VERIFY

This statement reads each specified sector from the designated disk platter to insure that information has been written conrrectly to these sectors.

```
VERIFY F (0,90)
```


PRINTING OUTPUT DEVICES

All printing output devices produce printed output from the following statements when the devices have been initially addressed (SELECTed) with a SELECT statement.

PRINT

Produces printed output on the output device in zoned or packed format.

```
10 PRINT A$,B,C,D(4)
20 PRINT "X="; X*75, "Y="; Y*100
```

PRINTUSING

Produces printed output formatted as indicated in the referenced Image (%) statement.

```
100 PRINTUSING 101, A$, B
101% ITEM ########## COST $##,###.##
```

HEXPRINT (not on 2200A)

Produces printed output in hexadecimal form. Alpha scalars or alpha array elements can be output either one value per line or as a continuous string of hex codes.

```
10 HEXPRINT A$; B$( )
20 HEXPRINT A$, B$( )
```

SELECT CO

Selects the output device to print all console output (CO); any characters keyed in on the keyboard, Immediate Mode PRINT or HEXPRINT statements, literals from INPUT statements, INPUT question mark, Ready-status colon, error codes, TRACE output, STEP mode output.

```
SELECT CO 215
```

NINE-TRACK TAPE DRIVE

An interface medium between a Wang system and a large scale computer (such as an IBM 360) or a backup medium for Wang disk units, the nine-track tape drive must have access to 12K user RAM and General I/O statements. These statements cannot be obtained on a 2200A, must be added to a 2200B or C with OP-2 and to a 2200S or WCS/10 with OP-23 or 24. Wang supported utilities to operate this unit are available.

$GIO

Provides a programmable I/O signal sequence to activate and control the unit.

$GIO REWIND /07B (6CF5 4400 8607, A$)

Tape operation sequences developed at Wang for operation of this unit are:

$GIO BSF /07B (6CF3 4400 8607, A$)

Backspaces one file to Tape Mark or BOT.

$GIO BSR /07B (6CF1 4400 8607, A$)

Backspaces one block (record) or to BOT.

$GIO BSW (6CF0 4400 8607, A$)

Backspaces one block and positions tape for writing.

$GIO CLEAN (6CF7 4400 8607, A$)

Backspaces tape over tape cleaner up to five times and repositions tape to reread.

$GIO FSF (6CF4 4400 8607, A$)

Spaces tape forward to next Tape Mark or EOF.

$GIO FSR (6CF2 4400 8607, A$)

Spaces tape forward one block (record).

$GIO READ (6CFB 4400 C222 8607, A$) B$( )

Reads a record (block).

$GIO REREAD (6CF8 4400 C222 8607, A$) B$( )

Backspaces one block and rereads the block just read; corrects single track errors.

$GIO REWIND (6CF5 4400 8607, A$)

Rewinds to load point (BOT).

$GIO WEOF (6CF9 4400 8607, A$)

Writes a Tape Mark (EOF).

$GIO WGAP (6CF6 4400 8607, A$)

Passes over 3.5 in. of tape to bypass a bad spot.

$GIO WRITE (6CFA 4400 A206 8607, A$) B$( )

Write a block (record).

When neither an address (e.g., 07B) nor a file number (e.g., #1) is specified, the device address currently selected for TAPE is used.

DIGITIZER

Five BASIC statements can be used to receive data from the Digitizer and store it in memory. The Digitizer must be selected or addressed with the appropriate SELECT statement.

INPUT

Accepts individual points from the Digitizer.

```
SELECT INPUT 65A
INPUT A, B, A$
```

DATALOAD*

Reads values into scalars or arrays. An alpha scalar or array element must contain at least 11 bytes.

```
SELECT TAPE 65A
DIM A(100)
DATALOAD A( )
```

DATALOAD BT*

Reads points into a variable or array.

```
SELECT TAPE 65A
DIM A (100) 12
DATALOAD BT /65A, A$( )
```

KEYIN (not on 2200A)

Polls the digitizer to determine if it is ready to transmit; if so, reads in the first character and branches to the referenced line number. The second line number is not used but must appear.

```
SELECT INPUT 65A
KEYIN A$, 100, 100
```

MAT INPUT (not on 2200A; on 2200B, C with OP-1; on 2200S, WCS/10 with OP-21)

Inputs moderately large amounts of data at relatively high speed; permits more efficient operation for large amounts of data than INPUT or KEYIN.

```
DIM A$ (100) 11
SELECT INPUT 65A
MAT INPUT A$
```

*not on 2200A, on 2200S or WCS/10 only with Advanced Programming Statement.

INTERFACE CONTROLLERS

These controllers have been designed to interface non-Wang devices to Wang systems. They can be obtained as input-only or input/output devices. The BASIC statements which activate the controllers are governed by the function of the device; an input-only device cannot use output-type statements. A controller must be selected or addressed with the appropriate SELECT statement.

```
SELECT INPUT 25A
```

When a device is used for INPUT, several statements can be used to activate it.

INPUT

Enables the controller for input of data to fill the variables specified. Each value must be terminated by an ASCII carriage return.

```
INPUT A, B, C, C$
```

KEYIN (not on 2200A)

Can be used to obtain single character input (any 8-bit code) and optimize scanning operations for several devices because of the capability to test 'ready' condition.

```
DIM A$1
KEYIN A$, 50, 60
```

DATALOAD BT*

Can be used with some external devices to obtain consecutive character input. Data rate significantly faster than INPUT. Can receive any 8-bit code; input is terminated by count or special character.

```
10 DIM A$ (5) 60
20 SELECT TAPE 23A
30 DATALOAD BT (N=300) A$( )
```

*Not on 2200A, on 2200S or WCS/10 only with Advanced Programming Statements.

MAT INPUT (not on 2200A; on 2200B or C with OP-1; on 2200S, WCS/10 with OP-21, 22, 23 or 24)

Enables device to receive high-speed input of alphanumeric data. Terminates one or more data elements with ASCII carriage return.

```
DIM A$(20)
MAT INPUT A$
```

$GIO (not on 2200A; on 2200B or C with OP-2; on 2200S, WCS/10 with OP-23 or 24)

Enables multi-readouts from I/O controllers with a programmable signal sequence. Provides great flexibility by controlling specific I/O operations.

```
DIM X$ (20) 12, A$10
$GIO/25A (C640, R$) X$( )
```

As an output device, a controller is activated by several output-type statements.

PRINT

Can output literals, data, hex codes.

```
PRINT HEX(22); A$, N, "METER DATA"
```

DATASAVE BT (not on 2200A; on 2200S, WCS/10 only with Advanced Programming Statements)

Outputs 8-bit data from any specified alphanumeric scalar or array.

```
DIM A$ (100)5
DATASAVE BT #5, A$( )
```

PRINTUSING

Outputs data from any specified scalar or array in the format specified in the referenced Image (%) statement.

```
100 PRINTUSING 101, "AMPS";A
101 %#### #####
```

$GIO (not on 2200A; on 2200B, C with OP-2; on 2200S, WCS/10 with OP-23 or 24)

Enables an I/O controller to be used for rapid data output with a programmable signal sequence.

```
10 DIM R$10, D$(20)60
100 $GIO WRITE /62B (6C01 4400 A206 8601,
    R$) D$( )
```


TC CONTROLLER

The TC Controller is used to facilitate asynchronous transmissions between a Wang CPU and some other system, either over a telephone line or over a direct line hook-up. Each TC unit must initially be accessed with the appropriate SELECT statement.

```
SELECT INPUT 219 (for input)
SELECT PRINT 21D (for output)
```

Statements which are normally used for input (reception) operations.

INPUT

Accepts data for scalars or array elements.

```
SELECT INPUT 219
INPUT A$, N, C$(1)
```

KEYIN (not on 2200A)

Accepts data one character at a time.

```
SELECT INPUT 219
DIM R$1
KEYIN R$, 100, 700
```

$GIO (not on 2200A; on 2200B, C with OP-2; on 2200S, WCS/10 with OP-23 or 24)

Enables the TC controller to accept data stream input from specified input device.

```
DIM B$10, C$(10)
$GIO INPUT/219 (FD01, B$) C$( )
```

The PRINT statement can be used for output (transmission) operations.

PRINT

Accepts data for scalars or array elements.

```
10 SELECT PRINT 21D
20 PRINT T$
```

$GIO (not on 2200A; on 2200B, C with OP-2; on 2200S, WCS/10 with OP-23 or 24)

Enables controller to receive character and echo it to specified output device.

```
10 DIM B$10, C$(20) 60
20 $GIO TC OUTPUT/21D (A403, B$) C$( )
```

ERROR MESSAGES

ERR 01 TEXT OVERFLOW
ERR 02 TABLE OVERFLOW
ERR 03 MATH ERROR
ERR 04 MISSING LEFT PARENTHESIS
ERR 05 MISSING RIGHT PARENTHESIS
ERR 06 MISSING EQUALS SIGN
ERR 07 MISSING QUOTATION MARKS
ERR 08 UNDEFINED FN FUNCTION
ERR 09 ILLEGAL FN USAGE
ERR 10 INCOMPLETE STATEMENT
ERR 11 MISSING LINE NUMBER OR CONTINUE ILLEGAL
ERR 12 MISSING STATEMENT TEXT
ERR 13 MISSING OR ILLEGAL INTEGER
ERR 14 MISSING RELATION OPERATOR
ERR 15 MISSING EXPRESSION
ERR 16 MISSING SCALAR
ERR 17 MISSING ARRAY
ERR 18 ILLEGAL VALUE
ERR 19 MISSING NUMBER
ERR 20 ILLEGAL NUMBER FORMAT
ERR 21 MISSING LETTER OR DIGIT
ERR 22 UNDEFINED ARRAY VARIABLE
ERR 23 NO PROGRAM STATEMENTS
ERR 24 ILLEGAL IMMEDIATE MODE STATEMENT
ERR 25 ILLEGAL FOR/NEXT USAGE
ERR 27 INSUFFICIENT DATA
ERR 28 DATA REFERENCE BEYOND LIMITS
ERR 29 ILLEGAL DATA FORMAT
ERR 30 ILLEGAL COMMON ASSIGNMENT
ERR 31 ILLEGAL LINE NUMBER
ERR 33 MISSING HEX DIGIT
ERR 34 TAPE READ ERROR
ERR 35 MISSING COMMA OR SEMICOLON
ERR 36 ILLEGAL IMAGE STATEMENT
ERR 37 STATEMENT NOT IMAGE STATEMENT
ERR 38 ILLEGAL FLOATING POINT FORMAT
ERR 39 MISSING LITERAL STRING

ERR 40 MISSING ALPHANUMERIC VARIABLE
ERR 41 ILLEGAL STR(ARGUMENTS
ERR 42 FILE NAME TOO LONG
ERR 43 WRONG VARIABLE TYPE
ERR 44 PROGRAM PROTECTED
ERR 45 STATEMENT LINE TOO LONG
ERR 46 NEW STARTING STATEMENT NUMBER TOO LOW
ERR 47 ILLEGAL OR UNDEFINED DEVICE SPECIFICATION
ERR 48 UNDEFINED KEYBOARD FUNCTION
ERR 49 END OF TAPE
ERR 50 PROTECTED TAPE
ERR 51 ILLEGAL STATEMENT
ERR 52 EXPECTED DATA (NONHEADER) RECORD
ERR 53 ILLEGAL USE OF HEX FUNCTION
ERR 54 ILLEGAL PLOT ARGUMENT
ERR 55 ILLEGAL BT ARGUMENT
ERR 56 NUMBER EXCEEDS IMAGE FORMAT
ERR 57 VALUE NOT BETWEEN 0 AND 32767
ERR 58 EXPECTED DATA RECORD
ERR 59 ILLEGAL ALPHA VARIABLE
ERR 60 ARRAY TOO SMALL
ERR 61 DISK HARDWARE ERROR
ERR 62 FILE FULL
ERR 63 MISSING ALPHA ARRAY DESIGNATOR
ERR 64 SECTOR NOT ON DISK
ERR 65 DISK HARDWARE MALFUNCTION
ERR 66 FORMAT KEY ENGAGED
ERR 67 DISK FORMAT ERROR
ERR 68 LRC ERROR
ERR 71 CANNOT FIND SECTOR
ERR 72 CYCLIC READ ERROR
ERR 73 ILLEGAL ALTERING OF A FILE
ERR 74 CATALOG END ERROR
ERR 75 COMMAND ONLY (NOT PROGRAMMABLE)
ERR 76 MISSING < OR > (PLOT ENCLOSURES)
ERR 77 STARTING SECTOR > ENDING SECTOR)

ERR 78	FILE NOT SCRATCHED
ERR 79	FILE ALREADY CATALOGED
ERR 80	FILE NOT IN CATALOG
ERR 81	/XXX DEVICE SPECIFICATION ILLEGAL
ERR 82	NO END OF FILE
ERR 83	DISK HARDWARE FAILURE
ERR 84	NOT ENOUGH MEMORY FOR MOVE OR COPY
ERR 85	READ AFTER WRITE ERROR
ERR 86	FILE NOT OPEN
ERR 87	COMMON VARIABLE REQUIRED
ERR 88	LIBRARY INDEX FULL
ERR 89	MATRIX NOT SQUARE
ERR 90	MATRIX OPERANDS NOT COMPATIBLE
ERR 91	ILLEGAL MATRIX OPERAND
ERR 92	ILLEGAL REDIMENSIONING OF ARRAY
ERR 93	SINGULAR MATRIX
ERR 94	MISSING ASTERISK
ERR 95	ILLEGAL MICROCOMMAND
ERR 96	MISSING ARG 3 BUFFER
ERR 97	VARIABLE OR ARRAY TOO SMALL
ERR 98	ILLEGAL ARRAY DELIMITERS
ERR =1	MISSING NUMERIC ARRAY NAME
ERR =2	ARRAY TOO LARGE
ERR =3	ILLEGAL DIMENSIONS

SYSTEM ERROR! Unrecoverable machine error or programmer faux pas (no execution phase).