

WANG

TO: Multiuser Basic-2 Language Reference Manual

FROM: Technical Publications Department

SUBJECT: Update to Multiuser Basic-2 Language Reference Manual
(700-4080E.02)

DATE: August 1989

This document updates Chapters 4, 5, 7, 11 and 12 of the Multiuser Basic-2 Language Reference Manual (700-4080E.02). It reflects new technical information for the CS386 CPU.

To update the Multiuser Basic-2 Language Reference Manual use the following collating instructions:

Remove

Chapter 4
Pages 4-14

Chapter 5
Pages 5-13, 5-27

Chapter 7
Pages 7-25, 7-26

Chapter 11
Pages 11-77 through 11-84
11-129 through 11-136

Chapter 12
Pages - None

Insert

Chapter 4
Pages 4-14, 4-15

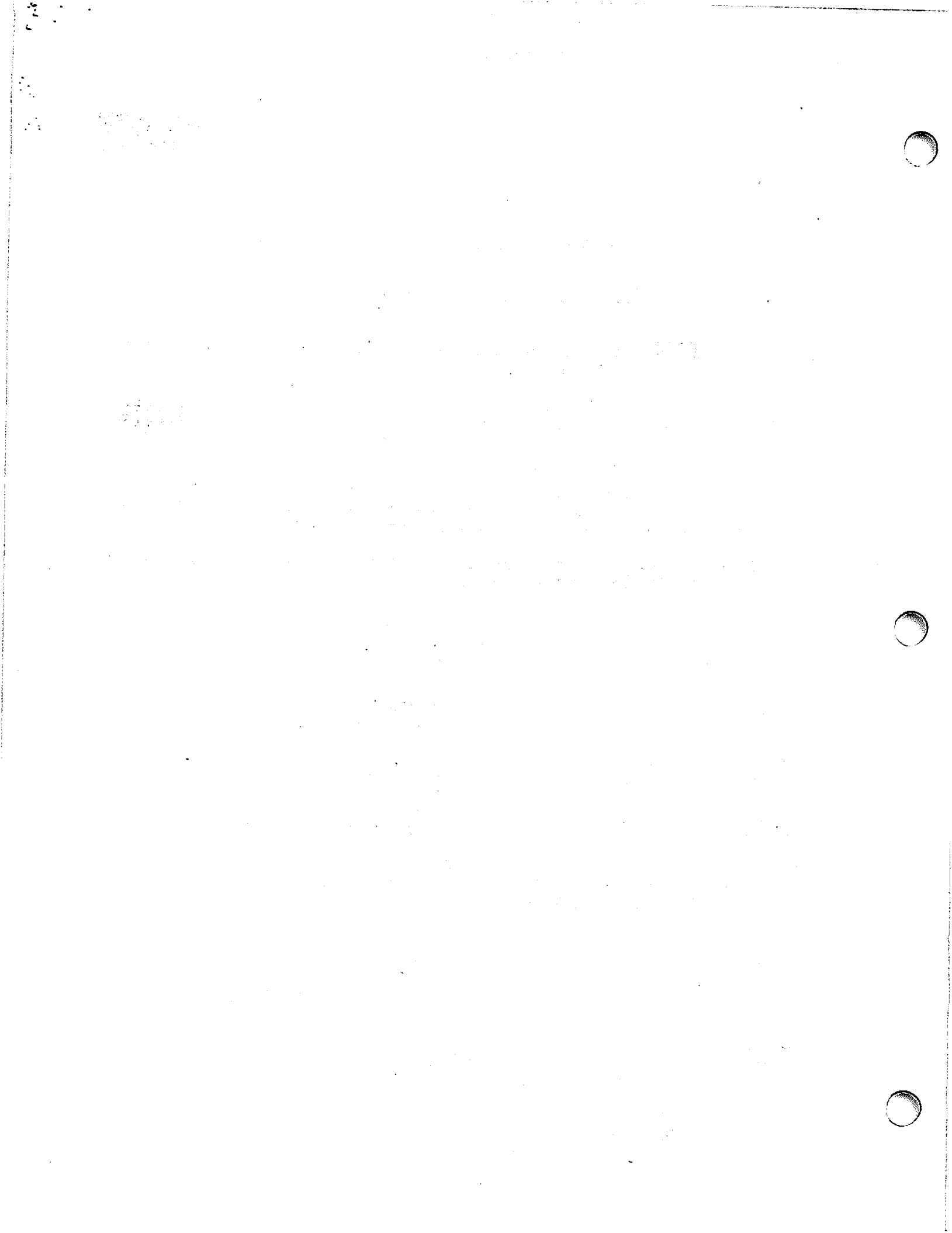
Chapter 5
Pages 5-13, 5-27

Chapter 7
Pages 7-25, 7-26

Chapter 11
Pages 11-77 through 11-84
11-129 through 11-136

Chapter 12
Pages 12-36a, 12-36b, 12-36c
12-36d, 12-36e, 12-36f
12-36g, 12-36h

Retain the original Customer Comment form, Order form, and back cover. Insert the new forms behind the appropriate original pages. You may discard the new back cover.



CS

Multiuser BASIC-2 Language Reference Manual

6th Edition — June 1987
Copyright © Wang Laboratories, Inc., 1987
700-4080E

2nd Update — August 1989
Copyright © Wang Laboratories, Inc., 1989
700-4080E.02

WANG

WANG LABORATORIES, INC.
ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 TEL. (508) 459-5000, TELEX 172108

Disclaimer of Warranties and Limitation of Liabilities

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual. However, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase, lease, or license agreement by which the product was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of the product, the accompanying manual, or any related materials.

Software Notice

All Wang Program Products (software) are licensed to customers in accordance with the terms and conditions of the Wang Standard Software License. No title or ownership of Wang software is transferred, and any use of the software beyond the terms of the aforesaid license, without the written authorization of Wang, is prohibited.

Warning

This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instructions manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device, pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference, in which case the user, at his own expense, will be required to take whatever measures may be required to correct the interference.

CONTENTS

PREFACE

CHAPTER 1 INTRODUCTION

Overview	1-1
Multiuser Operation	1-1
Interpretation Process	1-1
Editing Features	1-2
Compatibility Features	1-2
Types of BASIC-2 Instructions	1-2
System Commands	1-2
Statements	1-3
BASIC-2 Statements and Program Execution	1-3
Program Mode	1-3
Immediate Mode	1-4
Multiple-Statement Lines	1-5
Phases of the Language Processor	1-5
Entry Phase	1-6
Resolution Phase	1-6
Execution Phase	1-6

CHAPTER 2 EDITING AND DEBUGGING FEATURES

Overview	2-1
Line Entry	2-1
Spaces	2-1
Maximum Line Length	2-2
Upper/lowercase Entry	2-2
Line Editing	2-2
Edit Keys and Their Operation	2-3
Program Development	2-6
Replacing Lines	2-6
Deleting Lines	2-6
Renumbering Program Lines	2-7
Combining Program Lines	2-7

CONTENTS (continued)

Program Debugging Features	2-8
Descriptive Error Messages	2-9
Stopping and Resuming Program Execution	2-9
Halting and Stepping Through a Program	2-10
Tracing Through a Program	2-10
Listing and Cross-Referencing a Program	2-10
CHAPTER 3 SCREEN OPERATIONS	
Overview	3-1
Screen Control Codes	3-1
Cursor Control Codes	3-2
Character Display Attributes	3-3
HEX Codes Used to Invoke Display Attributes	3-3
Turning On Character Display Attributes	3-4
Turning Off Character Display Attributes	3-7
Using Isolated HEX(OE)	3-9
Special Uses of Alternate Display Attributes	3-11
Summary of Display Attribute Rules	3-12
Selection of Character Sets	3-13
Summary of Character Set Selection	3-14
Box Graphics	3-15
CHAPTER 4 NUMERIC OPERATIONS	
Overview	4-1
Numeric Values	4-1
Fixed-Point Format	4-2
Exponential Format	4-2
Numeric Constants	4-2
Numeric Variables	4-3
Numeric Expressions	4-4
Arithmetic Operators	4-4
Order of Evaluation	4-5
Altering the Standard Order of Evaluation	4-5
Round/Truncate Option	4-6
Computational Errors	4-6
System-Defined Numeric Functions	4-6
INT and FIX Functions	4-8
MAX and MIN Functions	4-9
MOD Function	4-10
RND (Random Number) Function	4-10
ROUND Function	4-11
SGN (Sign) Function	4-12
Trigonometric Functions	4-13

CONTENTS (continued)

Special-Purpose Numeric Functions	4-13
SPACE Function	4-13
SPACEK Function	4-13
SPACE S and SK	4-14
#ID Function	4-14

CHAPTER 5 ALPHANUMERIC STRING MANIPULATION INSTRUCTIONS

Alphanumeric Character Strings	5-1
Alphanumeric String Variables	5-1
Alphanumeric-Variable Length	5-2
STR Function	5-3
Alphanumeric Literal Strings	5-4
Hexadecimal Literal Strings	5-5
Concatenation of Strings	5-6
Using the Alpha-Array as a Scalar-Variable	5-7
Alphanumeric Expressions	5-8
General Forms of the Alphanumeric and Special-Purpose Functions and Operators	5-9
ALL Function	5-10
AND, OR, XOR Operators	5-11
BIN Function	5-13
BOOL Operator	5-14
DATE	5-16
HEX Literal	5-17
LEN Function	5-18
NUM Function	5-20
POS Function	5-22
STR Function	5-24
TIME	5-26
VAL Function	5-27
VER Function	5-28

CHAPTER 6 BINARY AND PACKED DECIMAL ARITHMETIC OPERATORS

Overview	6-1
Decimal to Binary Conversion and Two's Complement Notation	6-2
Decimal to Packed Decimal (BCD) Conversion and Ten's Complement Representation	6-4
General Forms of the Binary and Packed Decimal Operators	6-6
ADD Operator	6-7
DAC Operator	6-9
DSC Operator	6-10
SUB Operator	6-12

CONTENTS (continued)

CHAPTER 7 THE SELECT STATEMENT

Overview	7-1
Math Mode Selection	7-1
Specifying Degrees, Radians, or Grads	7-2
Selecting Rounding or Truncation	7-2
Computational Errors	7-2
Default Math Modes	7-3
Output Parameter Specification	7-3
Selecting a Pause	7-3
Selecting the Number of Output Lines	7-4
Selecting the Line Width	7-4
I/O Device Selection	7-6
The Classes of I/O Operations	7-6
Device-Addresses	7-6
Selecting Device-Addresses for I/O Operations	7-7
System Default Device-Addresses	7-8
The Device Table	7-9
Modifying Device Table Entries	7-10
Explicit Device Table Modification	7-10
The Console Input Select-Parameter	7-11
The INPUT Select-Parameter	7-11
The Console Output Select-Parameter	7-11
The PRINT Select-Parameter	7-12
The LIST Select-Parameter	7-13
The PLOT Select-Parameter	7-13
The TAPE Select-Parameter	7-13
The DISK Select-Parameter	7-14
The File-Number Select-Parameter	7-15
Multiple Select-Parameters in a Single SELECT Statement	7-16
Implicit Device Table Modification	7-16
Master Initialization	7-17
RESET	7-17
CLEAR and LOAD RUN	7-17
Device Types	7-18
Conditional Selection of Select-Parameters	7-19
General Forms of the SELECT Statements	7-20
LIST DT	7-21
ON/SELECT	7-23
SELECT Statement	7-25
SELECT Function	7-27

CHAPTER 8 PROGRAMMABLE INTERRUPTS

Overview	8-1
Interrupt Programming	8-1

CONTENTS (continued)

	Interrupt Processing	8-2
	Listing Interrupt Status	8-3
	Definition and Enabling of Interrupts	8-3
	General Forms of the Interrupt Control Instructions	8-4
	\$ALERT	8-5
	LIST I	8-6
	SELECT ON/OFF	8-8
	SELECT ON CLEAR	8-11
CHAPTER 9	ERROR CONTROL FEATURES	
	Overview	9-1
	Error Recoverability	9-2
	ERR Function	9-3
	ERR\$ Function	9-5
	ERROR	9-6
	SELECT ERROR	9-8
CHAPTER 10	SYSTEM COMMANDS	
	Overview	10-1
	General Forms of the System Commands	10-3
	CLEAR	10-4
	CONTINUE	10-6
	Function Keys	10-7
	HALT	10-8
	LIST	10-10
	LIST COM/DIM	10-14
	LIST T	10-15
	LIST V	10-17
	LIST #	10-20
	LIST '	10-22
	RENUMBER	10-23
	RESET	10-25
	RUN	10-26
	STOP	10-27
	TRACE	10-28
CHAPTER 11	GENERAL-PURPOSE BASIC-2 STATEMENTS	
	Overview	11-1
	General Forms of the General-Purpose Statements	11-5
	COM	11-6
	COM CLEAR	11-9

CONTENTS (continued)

CONVERT	11-11
DATA	11-15
DEFFN	11-17
DEFFN'	11-20
DIM	11-26
DO Group	11-29
END	11-30
FOR	11-31
\$FORMAT	11-35
GOSUB	11-36
GOSUB'	11-38
GOTO	11-40
HEXPACK	11-41
HEXUNPACK	11-44
IF...THEN	11-45
Image (%)	11-48
INPUT	11-49
KEYIN	11-52
LET	11-54
LINPUT	11-55
MAT COPY	11-58
MAT MOVE	11-60
MAT SEARCH	11-64
NEXT	11-68
ON GOTO/GOSUB	11-70
PACK	11-72
\$PACK	11-74
PRINT	11-84
PRINT AT	11-93
PRINT BOX	11-95
PRINT HEXOF	11-97
PRINT TAB	11-98
PRINTUSING	11-99
PRINTUSING TO	11-107
READ	11-109
REM	11-110
RESTORE	11-112
RETURN	11-114
RETURN CLEAR	11-116
ROTATE	11-117
STOP	11-119
\$STRAN	11-120
UNPACK	11-124
SUNPACK	11-125

CONTENTS (continued)

CHAPTER 12 DISK I/O STATEMENTS

Overview	12-1
File Catalog Mode	12-1
Initializing the Catalog	12-3
Saving Cataloged Programs on Disk:	
The SAVE Statement	12-3
Saving Modified Programs on Disk:	
The RESAVE Statement	12-4
Changing the Names of Files:	
The RENAME Statement	12-4
Retrieving Programs Stored on Disk:	
The LOAD Command	12-4
Listing the Catalog Index: The LIST DC Statement	12-5
The LOAD RUN Command	12-6
Saving Data Files	12-6
The DATASAVE DC OPEN Statement	12-7
The DATASAVE DC Statement	12-8
Opening a Second Data File on Disk	12-10
The DATALOAD DC OPEN Statement	12-11
The DATALOAD DC Statement	12-12
The DSKIP and DBACKSPACE Statements	12-13
Scratching Unwanted Files	12-15
Making Backup Copies of Cataloged Files	12-15
Closing a Data File	12-15
Sector Address Mode	12-16
RAMDISK	12-18
What is RAMDISK	12-18
Setting up RAMDISK	12-19
Accessing RAMDISK	12-19
General Forms of the Disk I/O Statements	12-19
COPY	12-21
DATALOAD BA	12-22
DATALOAD BM	12-23
DATALOAD DA	12-24
DATALOAD DC	12-26
DATALOAD DC OPEN	12-27
DATASAVE BA	12-28
DATASAVE BM	12-29
DATASAVE DA	12-30
DATASAVE DC	12-32
DATASAVE DC CLOSE	12-34
DATASAVE DC OPEN	12-35
DATALOAD AC	12-36a
DATALOAD AC OPEN	12-36b
DATASAVE AC	12-36c
DATASAVE AC OPEN	12-36d
DATASAVE AC CLOSE	12-36e
DATASAVE AC END	12-36f

CONTENTS (continued)

DBACKSPACE	12-37
DSKIP	12-38
\$FORMAT DISK	12-39
LIMITS	12-40
LIST DC	12-42
LOAD (Immediate mode)	12-45
LOAD (Program mode)	12-46
LOAD DA (Immediate mode)	12-48
LOAD DA (Statement)	12-49
LOAD RUN	12-51
MOVE (file)	12-52
MOVE (disk)	12-53
MOVE END	12-54
RENAME	12-55
RESAVE	12-56
SAVE	12-57
SAVE DA	12-60
SCRATCH	12-61
SCRATCH DISK	12-62
VERIFY	12-64

CHAPTER 13 MATH MATRIX STATEMENTS

Overview	13-1
Array Dimensioning	13-2
Array Redimensioning	13-2
Redimensioning Rules	13-3
General Forms of the Math Matrix Statements	13-3
MAT Addition: MAT +	13-4
MAT Constant: MAT CON	13-5
MAT Equality: MAT =	13-6
MAT Identity: MAT IDN	13-7
MAT INPUT	13-8
MAT Inverse: MAT INV	13-10
MAT Multiplication: MAT *	13-13
MAT PRINT	13-14
MAT READ	13-15
MAT Redimension: MAT REDIM	13-17
MAT Scalar Multiplication: MAT()*	13-19
MAT Subtraction: MAT -	13-20
MAT Transpose: MAT TRN	13-21
MAT Zero: MAT ZER	13-22

CHAPTER 14 SORT STATEMENTS

Overview	14-1
Sorting Numeric Data	14-2

CONTENTS (continued)

Representation of Array Subscripts in the	
Locator-Array	14-2
General Forms of the Sort Statements	14-3
MAT MERGE	14-4
MAT MOVE	14-13
MAT SORT	14-18
CHAPTER 15	GENERAL I/O STATEMENTS
Overview	15-1
Considerations for the Use of \$GIO	15-1
I/O Bus	15-2
Address Bus Strokes	15-3
Output Bus Strokes	15-3
Wait For Ready	15-4
Input Bus Strokes	15-4
General Forms of the General I/O Statements	15-4
\$IF ON/OFF	15-5
\$GIO	15-8
CHAPTER 16	MULTIUSER OPERATION
Overview	16-1
Functional Description	16-2
User Memory Allocation	16-5
Peripheral Allocation	16-9
Automatic Program Bootstrapping	16-10
Disabled Programming	16-11
Broadcast Message	16-11
Foreground and Background Processing	16-12
Background Terminal Printer Operation	16-14
Global Partitions	16-14
Global Program Text	16-15
Local Variables Referenced in a Global Partition	16-17
Nesting Global Subroutines	16-18
Some Programming Considerations for Global Program Text	16-19
Global Variables	16-20
Declaring Global Variables in a Nonglobal Partition	16-22
Using Global Variables for Task Control	16-22
Further Details on Global Partitions	16-24
Terminal Connect/Disconnect Detection	16-34
Multiuser Language Features	16-34
\$BREAK	16-36
\$CLOSE	16-38

CONTENTS (continued)

DEFFN @PART	16-39
\$DISCONNECT	16-41
\$INIT	16-45
\$MSG	16-49
\$OPEN	16-50
\$PSTAT	16-52
\$RELEASE PART	16-55
\$RELEASE TERMINAL	16-57
SELECT @PART	16-59
Programming Considerations	16-61
General Programming Considerations	16-61
Considerations for Time-Dependent Programs	16-61
I/O Operations	16-62
I/O Statement Restrictions	16-62
Default Disk Address	16-63
Special Features of the 2236MXE Controller	16-64

APPENDIX A KEY CODES AND CHARACTER SETS

APPENDIX B ERROR MESSAGES AND RECOVERY

APPENDIX C COMPATIBILITY WITH 2200 SERIES CPU's

APPENDIX D DEVICE ADDRESSES

APPENDIX E GLOSSARY

INDEX

FIGURES

Figure 3-1	Examples of Control Codes	3-3
Figure 3-2	The Display Attributes	3-5
Figure 3-3	Highlighting Key Words	3-6
Figure 3-4	Using HEX(0F)	3-7
Figure 3-5	Selecting Another Attribute	3-8
Figure 3-6	Testing Isolated HEX(0E)	3-10
Figure 3-7	Using Isolated HEX(0E)	3-11
Figure 3-8	Division of a Character Space	3-14
Figure 3-9	Box Graphic Line Placement Relative to Character Position	3-16
Figure 3-10	Box Graphic Line Placement Relative to Graphic Character Set	3-16
Figure 7-1	Sample Device Table Screen	7-9
Figure 12-1	The Catalog Index Listing	12-5
Figure 12-2	Logical Record Consisting of One Sector	12-9
Figure 12-3	Two One-Sector Logical Records	12-10
Figure 12-4	Logical Records in TEST-1	12-13
Figure 14-1	Simplified Merge Sequence	14-6
Figure 14-2	Control-Variable Prior to Beginning MAT MERGE	14-7
Figure 14-3	Control-Variable Following Termination of MAT MERGE Due to Empty Row	14-19
Figure 14-4	Simplified Sort Sequence	14-19
Figure 15-1	Schematic of Input and Output Strokes for the Model 2250 I/O Interface Controller	15-3
Figure 16-1	Memory Bank Organization	16-6
Figure 16-2	The Universal Global Area	16-7
Figure 16-3	A Multibank System Configuration	16-8
Figure 16-4	Two Partitions Accessing Global Program Text	16-16
Figure 16-5	Nesting Global Subroutines	16-18
Figure 16-6	Variable Table Entries in Global and Nonglobal Partitions for Global and Local Variables	16-21
Figure 16-7	Use of Text Pointer and Stack to Control Flow of Execution Following a Subroutine Call	16-26
Figure 16-8	The Pointer Table	16-27
Figure 16-9	Job Flow Between Originating Partitions and Global Partition	16-28
Figure 16-10	Pointer Table for Partition #2 Following Master Initialization	16-29
Figure A-1	The Default Font Character Set	A-4
Figure A-2	The Alternate Font Character Set	A-5

TABLES

Table 2-1	Editing Functions	2-3
Table 3-1	Cursor Controls	3-2
Table 4-1	System-Defined Numeric Functions	4-7
Table 4-2	Special-Purpose Numeric Functions	4-14
Table 5-1	BOOLh Logical Functions	5-15
Table 5-2	Format-Character Definitions	5-29
Table 7-1	Default Addresses for Primary I/O Devices	7-8
Table 9-1	SELECT ERROR Return Values	9-9
Table 10-1	BASIC-2 System Commands	10-1
Table 11-1	General-Purpose BASIC-2 Statements	11-1
Table 11-2	Binary Values for HEXPACK Characters	11-42
Table 11-3	Valid Field Specifications	11-77
Table 11-4	Valid Delimiter Specifications	11-126
Table 11-5	Valid Field Specifications	11-129
Table 12-1	Sector Address Mode Statements and Commands	12-18
Table 13-1	Matrix Operations	13-1
Table 14-1	Values of Sign Bits and Their Meanings	14-15
Table 14-2	Decimal and Decimal Complement Forms	14-16
Table 15-1	Legend	15-21
Table 15-2	Summary Microcommand Categories	15-23
Table 15-3	Single Address Strobe	15-24
Table 15-4	Control Microcommands	15-25
Table 15-5	Single Character Output Microcommands	15-27
Table 15-6	Single Character Input Microcommands	15-29
Table 15-7	Multicharacter Output Microcommands	15-30
Table 15-8	Valid CHECK T Codes for Table 15-7	15-31
Table 15-9	Valid LEND Codes for Table 15-7	15-31
Table 15-10	Multicharacter Input Microcommands	15-32
Table 15-11	Valid CHECK T Codes for Table 15-10	15-33
Table 15-12	Valid LEND Codes for Table 15-10	15-34
Table 15-13	Register Usage	15-35
Table 16-1	Functions of Pointer Table Items	16-31
Table 16-2	Statements Which Modify the Pointer Table	16-33
Table 16-3	Multiuser Functions	16-35
Table 16-4	Devices to Which BASIC-2 Statements Communicate	16-63
Table A-1	BASIC-2 Standard Key Codes	A-1
Table A-2	BASIC-2 Special Key Codes	A-2

PREFACE

This manual is designed as a primary resource for using the BASIC-2 language on Wang computer systems. Users unfamiliar with the BASIC language are encouraged to refer to a standard textbook for an introduction to the language.

Throughout this manual, a general format accompanies each description of a command or statement. When more than one specific arrangement is permitted, there are separate numbered formats. Within a format, key words, connectives, and special characters appear in proper sequence. Unless otherwise stated, you can use only the sequence shown.

This manual uses the following conventions to define and illustrate the components of BASIC-2 program statements and commands:

- Uppercase letters (A through Z), digits (0 through 9), and special characters (such as *, /, +) must always be used for program entry exactly as presented in the general format.
- All lowercase words represent information that you must supply.

Example:

In the following statement, you must supply the line-number.

GOTO line-number

- When braces, à è enclose a vertically stacked list in a portion of a format, you must select one of the options within the braces.

Example:

```
    expression
ON
    alpha-variable
```

- Brackets, [] indicate that the enclosed information is optional. When brackets contain a vertical list of two or more items, you can use one or none of the items.

Example:

LOAD RUN [filename]

- The presence of an ellipsis (...) within any format indicates that the unit immediately preceding the notation can occur one or more times in succession.

Example:

COM com-element [,com-element] ...

- When one or more items appear in sequence, these items or their replacements must appear in the specified order.

Trigonometric Functions

The trigonometric functions SIN, COS, and TAN and their inverse functions, ARCSIN, ARCCOS, and ARCTAN, can be calculated in one of three modes: radians, degrees, or grads (360 degrees = 400 grads). Trigonometric functions are evaluated in radians, unless the system is explicitly instructed to use degrees or grads. If degrees or grads are required, they must be specified with the following SELECT statements prior to performing trigonometric calculations:

SELECT D -- Use degrees in all subsequent trigonometric calculations.

SELECT G -- Use grads in all subsequent trigonometric calculations.

SELECT R -- Use radians in all subsequent trigonometric calculations.

Radian measure is automatically selected upon Master Initializing the system or when a CLEAR command is issued.

SPECIAL-PURPOSE NUMERIC FUNCTIONS

A second group of numeric functions is available for certain special-purpose operations. These functions are summarized in Table 4-2. With the exceptions of the #ID, ERR, SPACE, and SPACEK functions, the remaining special-purpose numeric functions operate on alphanumeric arguments and are described in detail in Chapter 5. The ERR function is discussed with the error control features in Chapter 9.

SPACE Function

The SPACE function returns the amount of memory not currently occupied by program text or data, minus the amount of memory occupied by the value stack. The value returned represents the actual amount of free space in memory at any point during execution.

The Value Stack initially occupies zero bytes but expands during program execution. To determine how much free space is actually available, check the value of SPACE during program execution when the Value Stack attains its maximum size. Typically, the value stack reaches maximum size when the program executes the innermost loop in a series of nested loops.

SPACEK Function

Before memory has been partitioned, the SPACEK function returns the total amount of available user memory divided by 1,024. For example, a system with 56K of user memory returns SPACEK = 56. After a system has been partitioned, SPACEK returns the size of the partition that executes the SPACEK function.

SPACE S and SK

SPACE S determines the amount of memory that is not currently occupied by any partition. This is Ramdisk Memory. SPACE SK returns the total amount of memory including all allocated and Ramdisk memory. This is the total memory on the CPU board.

#ID Function

The #ID function returns the value of the CPU identification number (a number from 1 to 65535). With the #ID function, a program can distinguish one system from another. This capability is useful in licensing software to specific installations.

Table 4-2. Special-Purpose Numeric Functions

Function	Meaning	Examples
BIN	Converts an integer value to a binary number.	A\$ = BIN(65)
ERR	Returns the error code of the last error condition.	X = ERR
LEN	Determines the length of a character string.	X = LEN(A\$)
NUM	Determines whether or not a character string is a legal representation of a BASIC number.	X = NUM(A\$).
POS	Returns the position of the first (or last) character in a character string that meets a specified condition.	X = POS(A\$="\$")
VAL	Computes the decimal equivalent of a binary value.	X = VAL(A\$)
VER	Verifies that a character string conforms to a specified format.	Y = VER(B\$,"###")

(continued)

Table 4-2. Special-Purpose Numeric Functions (continued)

Function	Meaning	Examples
SPACE	Determines the amount of free space available in memory.	Z = SPACE
SPACE K	Returns the total user memory size or partition size divided by 1,024.	Z1 = SPACE K
SPACE S*	Determines the amount of memory that is not currently occupied by any partition (Ramdisk Memory).	Z2 = SPACE S
SPACE SK*	Returns the total amount of memory including all allocated and Ramdisk memory.	Z3 = SPACE SK
#ID	Returns the CPU identification number	PRINT #ID
<p>Note: * CS/386 ONLY</p>		



BIN Function

Format:

BIN(expression [,length])

where:

length= numeric-variable or the digit 1, 2, 3, or 4

If length= 1, 0 <= value-of-expression < 256

If length= 2, 0 <= value-of-expression < 65,536

If length= 3, 0 <= value-of-expression < 16,777,216

If length= 4, 0 <= value-of-expression < 4,294,967,296

BIN is an alphanumeric function that uses a numeric argument, but returns an alphanumeric value; it is the inverse of the VAL function. The BIN (binary) function converts the integer value of the expression to a binary value. The number of bytes in the binary value is specified by the digit; if no digit is included, the length is assumed to be one byte. A numeric-variable can now be used to specify the length of the binary result of the BIN function. The BIN function can only be used in the alpha-expression portion of an alphanumeric assignment statement. BIN is especially useful for code conversion and conversion of numbers from internal decimal format to binary.

Example:

Sets A\$ = A since the binary value of decimal 65 is the character code for the letter A.

A\$ = BIN(65)

Examples of valid syntax:

B\$ = BIN(X,L)

A\$ = BIN(X)

STR(A\$,I,2) = BIN(X,2)

C\$ = BIN(X*Y/Z,4)

BOOL Operator

Format:

BOOL h

where:

h = hexadecimal digit (0-9 or A-F)

BOOL is a generalized logical operator that performs a specified operation on the value of the receiver-variable. The operation to be performed is specified by the hexadecimal digit following BOOL (refer to Table 5-3). BOOL can be used only in the alpha-expression portion of an alpha assignment statement. (Refer to the discussion of alpha expressions and the alpha assignment statement in the section entitled "Alphanumeric Expressions".) The value of the operand and the value of the receiver-variable are operated upon, and the result is assigned to the receiver-variable. For example, the following statement logically not-ANDs the value of B\$ with the value of A\$ and assigns the result to A\$:

```
A$ = BOOL7 B$
```

The logical operations are performed on a character-by-character basis from left to right, starting with the leftmost character in each field.

- If the defined length of the operand is shorter than that of the receiver-variable, the remaining bytes of the receiver-variable are not changed.
- If the defined length of the operand is equal to that of the receiver-variable, the entire values of both, including any trailing spaces, are operated on. (Trailing spaces usually are not considered part of the value of an alpha-variable.)
- If the operand is longer than the receiver-variable, the operation terminates when the last byte of the receiver-variable has been operated on.

A specified portion of an alpha-variable can be operated on if the portion is defined with a string function. For example, the following statement operates only on the third and fourth bytes of A\$:

```
STR(A$,3,2) = BOOL9 B$
```


VAL Function

Format:

$$\text{VAL}(\left. \begin{array}{l} \text{\{alpha-variable\}} \\ \text{\{literal-string\}} \end{array} \right\} [,\text{length}])$$

where:

length = numeric-variable or the digit 1, 2, 3, or 4

VAL is a numeric function that uses an alphanumeric argument, but returns a numeric value; it is the inverse of the BIN function. The VAL (value) function converts the binary value in the alpha-variable or literal-string to a numeric value. The number of bytes in the binary value to be converted is specified by the digit; if no digit is included, the length is assumed to be one byte. A numeric-variable can now be used to specify the length of the binary value in the VAL function. The VAL function can be used wherever numeric functions are legal.

VAL is particularly useful in code conversion and table lookup operations since the converted number can be used as a subscript to retrieve the corresponding code from an array. Additionally, VAL can be used with the RESTORE statement to retrieve codes or data from DATA statements.

Examples:

```
:PRINT VAL(HEX(20))  
32
```

```
:A$=HEX(1234)  
:PRINT VAL(A$,2)  
4660
```

Examples of valid syntax:

```
X = VAL(A$,L)  
X=VAL(A$)  
PRINT VAL("A")  
Y=VAL(B$,2)  
RESTORE VAL(STR(A$,I,1))+1  
B$=A$(VAL(C$)+1)  
IF VAL(X$,2) > 1024 THEN 100
```

VER Function

Format:

$$\text{VER} \left(\begin{array}{l} \text{alpha-variable} \\ \text{literal-string} \end{array} \right) , \text{format-specification}$$

where:

$$\text{format-specification} = \begin{array}{l} \text{alpha-variable} \\ \text{literal-string} \end{array}$$

The VER (verify) function verifies that the value of an alphanumeric-variable or literal string conforms to a specified format. The first variable or literal string in the function is verified against the format specified by the second variable or literal string (the format-specification). The VER function returns the number of successive characters in the value being verified that conform to the format-specification. Each character in the defined length of the alpha-variable or literal string is verified by checking it against the character set associated with the specified format-character in the format-specification (refer to Table 5-4). If a character in the value being verified does not appear in the specified format-character set, it is regarded as an illegal character and causes a termination of the VER operation.

The verify operation terminates when an illegal character is found, when the end of the value (including any trailing spaces) is encountered, or when the end of the format-specification is reached.

VER is a special-purpose numeric function that uses an alphanumeric argument but returns a numeric result. The VER function can be used wherever numeric functions are legal. Refer to the discussion of numeric functions in Chapter 4.

SELECT Statement

Format:

```
SELECT select-parameter [,select-parameter ]...
```

where:

select-parameter = {
D
R
G
ERROR [> error-code]
[NO] ROUND
P [digit]
LINE numeric-expression
CI device-address
INPUT device-address
CO device-address [(width)]
PRINT device-address [(width)]
LIST device-address [(width)]
PLOT device-address
TAPE device-address
DISK device-address
file-number device-address
TC port-number
TERMINAL port-number
DRIVER device-address [OFF]
NEW *
OLD *

device-address = { /taa,
< alpha-variable > }

where:

t = one hex digit specifying the device-type
aa = two hex digits specifying the physical device address
alpha-variable = three-byte variable whose value must be three ASCII hex digits representing the device type and address
width = an expression 0 < 256 specifying the maximum number of characters on a single line
file-number = #n, where n = an integer or numeric-variable with a value >= 0 and < 16

Note: * CS/386 ONLY

The SELECT statement is used for the following purposes:

- To select the desired math modes for arithmetic operations; including type of measure for trigonometric functions, rounding or truncation of numeric results, and desired system response to specific math errors. (Refer to the section entitled "Math Mode Selection".)
- To select output parameters for communicating with output devices. (Refer to the section entitled "Output Parameter Specification".)
- To select device-addresses for accessing specified devices with input/output statements and commands. (Refer to Section 7.4.)
- To select a 2236MXE port for telecommunications. Refer to the *Asynchronous Communications User Guide for Model 2236MXE Terminal Processor and Option-W Terminal Processor (700-8098)* for a discussion of SELECT TC, SELECT TERMINAL, and telecommunications using the 2236MXE.
- To control printer drivers. Refer to the *BASIC-2 Utilities Manual* for a discussion the use of SELECT DRIVER for controlling printer drivers.
- To Select the program saving format mode. The option OLD (Default) signifies format compatible with the current 2200 & VLSI CPUs. The NEW option is only compatible with the CS/386 CPU. It should be noted that the CS/386 takes less processing time to resolve the NEW file format. Because the new format takes more space to save a program line, old formats being resaved in the new format may fail with an A05 Error. To overcome this situation, the program line must be broken into additional line numbers.

The alpha-variable or literal string following the F= parameter contains the field specifications for the buffer. Each field specification consists of two bytes. The first byte specifies the type of field; the second byte specifies the field width (i.e., the number of characters in that field).

Example:

The following two examples illustrate that the field specifications for a buffer can either be contained within an alphanumeric-variable or expressed as a hexadecimal literal string:

```
$PACK ( F = F$) B$( ) FROM X( )
$PACK ( F = HEX(1008)) B$( ) FROM X( )
```

If the first byte of the field specification is HEX(00), the corresponding field in the buffer is skipped. Alphanumeric fields are indicated by specifying HEX(A0) as the first byte of the field specification. Several types of numeric fields are permitted; numeric data is indicated by specifying a hex digit from 1 to 6 as the first hex digit of the first byte in the field specification. Each of the digits 1 to 6 identifies a unique numeric format. Refer to Table 11-3. The second hex digit specifies the implied decimal position in binary; the decimal point is assumed to be the specified number of digits from the right-hand side of the field. For example, the value +123.45 is stored as +12345, and the implied decimal point position is 2. An error results if a numeric value is packed into an alphanumeric field or if an alphanumeric value is packed into a numeric field.

Table 11-3. Valid Field Specifications

Numeric Fields	Meaning
00xx	skip field
10xx	ASCII free format
2dxx	ASCII integer format
3dxx	IBM display format
4dxx	IBM USASCII -- 8 format
5dxx	IBM packed decimal format
6dxx	unsigned packed decimal format
7d0y	packed decimal with binary overflow format
8d0y	signed binary format
9d0y	unsigned binary format
A0xx	alphanumeric field
Alxx	compressed alphanumeric format
<p>where:</p> <p>xx = field width in binary (xx > 0)</p> <p>y = field width in binary (0 < y < =4)</p> <p>d = implied decimal position in binary</p>	

You must supply a separate field specification for every variable or array in the variable list. All elements in an array use the field specification specified for that array.

Example:

The following statement requires three field specifications:

```
$PACK (F = F$) B$( ) FROM A$, B( ), C$
```

If F\$ = HEX(A0081006A010), then

A008 is the field specification for A\$

1006 is the field specification for each element in the array B()

A010 is the field specification for C\$

You can also mnemonically define a field specification in a \$FORMAT statement. \$FORMAT permits the use of simple mnemonic codes rather than hex codes to specify field formats. Refer to the discussion of the \$FORMAT statement in this section.

Example:

The field specifications defined for F\$ above could be defined as follows in a \$FORMAT statement

```
$FORMAT F$ = A8, F6, A16
```

Buffer format:

field 1	field 2	field 3	...	field n
---------	---------	---------	-----	---------

Data format:

- Alphanumeric Fields (A0xx)

C	C	...	C
---	---	-----	---

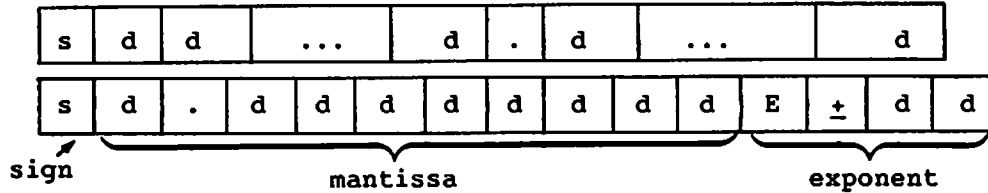
where:

C = one character of the value to be packed

If the value is shorter than the length of the field, the value is left-justified in the field and the remainder of the field is filled with spaces. If the value is too long, it is truncated to fit within the field.

• Numeric Fields

ASCII free format (10xx)



where:

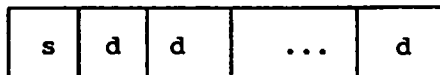
- s = sign (space if value ≥ 0 or minus sign (-) if value < 0)
- d = ASCII digit

Format 1 is used if the value is greater than or equal to 10^{-1} and less than 10^{+13} or if the value is less than 10^{-1} but can be expressed in fewer than 13 digits. Format 2 is used in all other cases.

Numeric values are stored as ASCII characters in one of the formats illustrated above. Note that Formats 1 and 2 are the same formats used by the PRINT statement when printing numeric values. If the value to be stored is shorter than the length of the field, the value is left-justified in the field and the remainder of the field is filled with spaces. If the value is too large to fit in the field, it is truncated to the length of the field.

For the numeric formats illustrated, the following rule applies: If the value is shorter than the length of the field, leading zeros are inserted; if the value is too long, an error results.

• ASCII integer format (2dxx)



where:

- s = sign (ASCII + or -), required
- d = ASCII digit

- IBM display format (3dxx)

Fd	Fd	...	Fd	sd
----	----	-----	----	----

where:

s = sign (C = +, D = -)
d = digit (0-9)

- IBM USASCII-8 format (4dxx)

5d	5d	...	5d	sd
----	----	-----	----	----

where:

s = sign (A = +, B = -)
d = digit (0-9)

- IBM packed decimal format (5dxx)

dd	dd	...	dd	ds
----	----	-----	----	----

where:

s = sign (C = +, D = -)
d = digit (0-9)

- Unsigned packed decimal format (6dxx)

dd	dd	...	dd	dd
----	----	-----	----	----

where:

d = digit (0-9)

The above formats are shown in hexadecimal notation.

Decimal arithmetic can be performed on unsigned packed decimal numbers. (Refer to the discussion of the DAC and DSC operators in Chapter 7.)

Example:

The following example assumes that the buffer B\$ has three fields (one alpha and two numeric), each five characters long. \$PACK packs the values of A\$, X, and Y into B\$.

```
:10 DIM B$15
:20 A$ = "ABC" :X = -12 :Y = +1.2345
:30 F$ = HEX(A00520051005)
:40 $PACK (F = F$) B$ FROM A$, X, Y
:50 PRINT "B$ = "; B$
:RUN
```

B\$ = "ABC -0012 1.23"

Example:

The following examples assume that X = 12.345.

<u>Statements</u>	<u>Results</u>
:10 F\$ = HEX(100A) :20 \$PACK (F = F\$) B\$ FROM X	B\$ = "12.345"
:10 F\$ = HEX(240A) :20 \$PACK (F = F\$) B\$ FROM X	B\$ = "+000123450" Note that there is an implied decimal point four digits from the right end of the field.
:10 F\$ = HEX(3305) :20 \$PACK (F = F\$) B\$ FROM X	B\$ = HEX(F1F2F3F4C5)
:10 F\$ = HEX(4206) :20 \$PACK (F = F\$) B\$ FROM X	B\$ = HEX(5050515253A4)
:10 F\$ = HEX(5506) :20 \$PACK (F = F\$) B\$ FROM X	B\$ = HEX(00001234500C)
:10 F\$ = HEX(2304) :20 \$PACK (F = F\$) B\$ FROM X	Results in an error because the receiving field is too small to hold the value.

• Packed Decimal with Binary Overflow Format (7d0y)

The Packed Decimal with Binary Overflow Format is used to pack numeric values. The number is stored in packed decimal format (same as type 5dxx) if it will fit in the specified field. The maximum field length allowed is 4. If the number is too large to be stored in packed decimal, it is converted to binary and stored in a binary format. If the binary number is still too large for the field, an error (ERROR X71) results. The last hexdigit of the packed value identifies the value as being either packed decimal or binary. If the last hexdigit is hex(C-F), the value is packed decimal. If the value is hex(0-B), the value is binary.

When a number is stored in binary format, it is first converted to a binary value the same length as the field. The number is too large to be packed if the upper hexdigit is greater than 5. The number is then shifted left by one hexdigit (4-bits). The low 3-bits of what was the upper hexdigit now become the upper 3-bits of the low hexdigit of the value. The lowest bit of the last hexdigit stores the sign of the value: zero for nonnegative and one for negative values.

Example:

```
10 X = 1234567
20 $PACK (F=HEX(7004)) D$ FROM X
30 $PACK (F=HEX(7003)) B$ FROM X
```

Results in D\$ = HEX(12 34 56 7C) and
B\$ = HEX(2D 68 72)

- Signed Binary Format (8d0y)

The Signed Binary Format is used to pack numeric values. The maximum field length allowed is 4. The value is converted to a binary value the same length as the field. If the binary number is too large for the field, an error (ERROR X71) results. Negative values are stored in 2's complement. Thus, the highest bit in the field can be used to determine the sign of the value: the bit is zero for nonnegative values and one for negative values.

Example:

```
10 X = 1234567
20 $PACK (F=HEX(8004)) P$ FROM X
30 $PACK (F=HEX(8004)) N$ FROM -X
```

Results in P\$ = HEX(00 2D 68 72) and
N\$ = HEX(FF D2 97 8E)

- Unsigned Binary Format (9d0y)

The Unsigned Binary Format is used to pack numeric values. The maximum field length allowed is 4. The sign of the number is ignored. The absolute value is converted to a binary value the same length as the field. If the binary number is too large for the field, an error (ERROR X71) results.

Example:

```
10 X = 1234567
20 $PACK (F=HEX(8004)) P$ FROM X
30 $PACK (F=HEX(8004)) N$ FROM -X
```

Results in P\$ = HEX(00 2D 68 72) and
N\$ = HEX(00 2D 68 72)

- Compressed Alphanumeric Format (Alxx)

The Compressed Alphanumeric Format provides a means to more compactly store characters with ASCII values hex(20) through hex(5F). These include the uppercase characters, digits, space, and certain symbols. Other characters in the string to be packed will cause an error (ERROR X71). The characters in the string to be packed are converted to 6-bit values. Specifically, characters hex(20) through hex(5F) are converted to the 6-bit values hex(00) through hex(3F). Then, the 6-bit characters are stored left-justified in the pack field. Thus, each four characters in the string to be packed is stored as three bytes in the pack field.

If the compressed value is shorter than the length of the field, the value is left-justified in the field and the remainder of the field is filled with 0 bits (effectively, the original value is padded with space characters on the right). If the compressed value is too long, it is truncated to fit within the field.

Example:

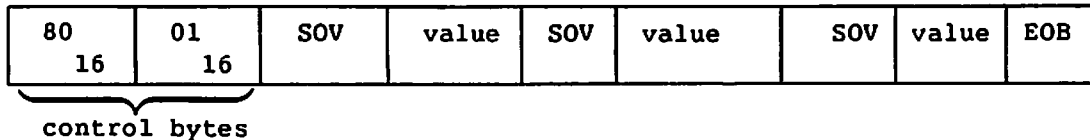
```
10 S$="ABCDEFGH"
20 $PACK (F=HEX(A103)) P$ FROM S$
```

Results in P\$ = HEX(86 28 E4)

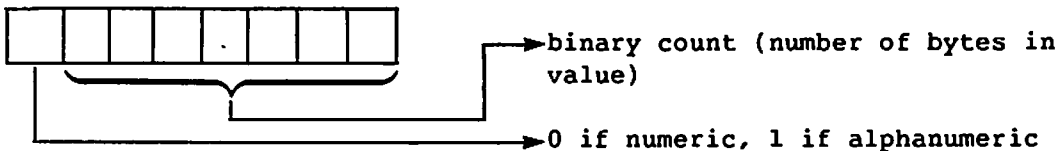
The Internal Form of the \$PACK Statement

The internal form of \$PACK stores data in the standard Wang 2200 disk record format. The values of the variables and arrays in the variable list are sequentially packed into the buffer. The packing terminates when all values have been packed.

Standard Wang 2200 Record Format (buffer format)



The SOV (Start-Of-Value) character precedes each data value in the record and indicates whether the value is numeric or alphanumeric and the length of the value.



The EOB (End-Of-Block, HEX(FD)) character indicates the end of valid data in the record.

Data format:

Alphanumeric Values



where:

C = any character of the alphanumeric value to be packed

Trailing spaces are included after the actual value so that the length of the entire value stored is the same as the defined length of the alphanumeric-variable.

PRINT

Format:

$$\text{PRINT [print-element] } \left[\left\{ \begin{array}{l} ' \\ ; \end{array} \right\} \text{ [print-element] } \right] \dots$$

where:

$$\text{print-element} = \left\{ \begin{array}{l} \text{alpha-variable} \\ \text{literal-string} \\ \text{numeric-expression} \\ \text{AT function} \\ \text{BOX function} \\ \text{HEXOF function} \\ \text{TAB function} \end{array} \right\}$$

The PRINT statement prints the values of the specified print-element(s) on a designated output device in a system-defined format. The PRINT statement can contain alphanumeric and numeric print-elements, as well as the PRINT functions AT, BOX, HEXOF, and TAB. These functions are described under separate headings later in this chapter.

In Program mode, PRINT outputs to the output device currently selected for PRINT operations. In Immediate mode, PRINT outputs to the currently selected Console Output device. The use of the Console Output device for Immediate mode PRINT output is a debugging feature that enables you to halt program execution and examine the results of Immediate mode PRINT statements on the screen while programmed PRINT output is selected to a printer. (Refer to Chapter 8 for a discussion of selecting PRINT and CO devices.)

Alphanumeric Print-Elements

An alphanumeric print-element can be a literal string, a HEX literal, or an alpha-variable. A literal string is printed exactly as it appears within the quotation marks, including trailing spaces. The quotation marks are not printed.

Example:

```
:10 PRINT "ABCD"  
:RUN  
ABCD
```

Example:

The following examples illustrate that the field specifications for a buffer can be either contained within an alphanumeric-variable or expressed as a hexadecimal literal string:

```
$UNPACK (F = F$) B$( ) TO X, Y, Z
```

```
$UNPACK (F = HEX(1008)) B$( ) TO X, Y, Z
```

If the first byte of the field specification is HEX(00), the corresponding field in the buffer is skipped. Alphanumeric fields are indicated by specifying HEX(A0) as the first byte of the field specification. Several types of numeric fields are permitted; numeric data is indicated by specifying a hex digit from 1 to 6 as the first hex digit of the first byte in the field specification. Each of the digits 1 to 6 identifies a unique numeric format. (Refer to Table 11-5.) The second digit specifies the implied decimal position in binary; the decimal point is assumed to be the specified number of digits from the right-hand side of the field. For example, if a field contains the value +12345 and an implied decimal position of 2 is specified, the value unpacked would be +123.45. An error results if a numeric field is unpacked into an alphanumeric-variable or if an alphanumeric field is unpacked into a numeric-variable.

Table 11-5. Valid Field Specifications

Numeric Fields	Meaning
00xx 10xx 2dxx 3dxx 4dxx 5dxx 6dxx 7d0y 8d0y 9d0y A0xx Alxx	skip field ASCII free format ASCII integer format IBM display format IBM USASCII - 8 format IBM packed decimal format unsigned packed decimal format packed decimal with binary overflow format signed binary format unsigned binary format alphanumeric field compressed alphanumeric format
<p>where:</p> <p>xx = field width in binary (xx > 0) y = field width in binary (< y <=4) d = implied decimal position in binary</p>	

You must supply a separate field specification for every variable or array in the variable list. All elements in an array use the field specification for that array.

Example:

The following statement requires three field specifications:

```
$UNPACK (F = F$) B$( ) TO A$, B( ), C$
```

If F\$ = HEX(A0081006A010), then

A008 is the field specification for A\$

1006 is the field specification for each element in the array B()

A010 is the field specification for C\$

You can also mnemonically define a field specification in a \$FORMAT statement. \$FORMAT permits the use of simple mnemonics rather than hex codes to specify field formats. Refer to the discussion of the \$FORMAT statement in this section.

Example:

The field specification defined for F\$ above could be defined as follows in a \$FORMAT statement:

```
$FORMAT F$ = A8, F6, A16
```

Buffer format:

field 1	field 2	field 3	...	field n
---------	---------	---------	-----	---------

Data format:

- Alphanumeric Fields (A0xx)

C	C	...	C
---	---	-----	---

where:

C = any character in an alphanumeric field to be unpacked.

- IBM USASCII-8 format (4dxx)

5d	5d	...	5d	sd
----	----	-----	----	----

where:

s = sign (A = +, B = -)*
d = digit (0-9)

- IBM packed decimal format (5dxx)

dd	dd	...	ds
----	----	-----	----

where:

s = sign (C = +, D = -)*
d = digit (0-9)

*The \$UNPACK statement considers B or D to be minus (-); any other hex digit is considered to be plus (+).

- Unsigned packed decimal format (6dxx)

dd	dd	...	dd
----	----	-----	----

where:

d = digit (0-9)

The above formats are shown in hexadecimal notation.

Decimal addition and subtraction can be performed on unsigned packed decimal numbers. (Refer to the discussion of the DAC and DSC operators in Chapter 6.)

Examples:

The following examples assume that B\$ = "+12345678901234567890".

Statements

Results

:100 F\$ = HEX(A005)	
:110 \$UNPACK (F = F\$) B\$ TO A\$	A\$ = "+1234"
:100 F\$ = HEX(1010)	
:110 \$UNPACK (F = F\$) B\$ TO X	Results in an error because B\$ contains more than 13 digits.
:100 F\$ = HEX(2015)	
:110 \$UNPACK (F = F\$) B\$ TO X	X = 1.234567890123E19

<u>Statements</u>	<u>Results</u>
:100 F\$ = HEX(2206)	
:110 \$UNPACK (F = F\$) B\$ TO X	X = 123.45
:10 B\$ = HEX(F1F2F3D4)	
:20 F\$ = HEX(3304)	
:30 \$UNPACK (F = F\$) B\$ TO X	X = -1.234
:10 B\$ = HEX(51525354A5)	
:20 F\$ = HEX(4005)	
:30 \$UNPACK (F = F\$) B\$ TO X	X = 12345
:10 B\$ = HEX(000012345C)	
:20 F\$ = HEX(5105)	
:30 \$UNPACK (F = F\$) B\$ TO X	X = 1234.5

• Packed Decimal with Binary Overflow Format (7d0y)

The Packed Decimal with Binary Overflow Format is used to unpack numeric values that were stored with \$PACK format 7d0y. The maximum field length allowed is 4. The last hexdigit of the packed value identifies the value as being either packed decimal or binary. If the last hexdigit is hex(C-F), the value is packed decimal (same as format 5dxx). If the value is hex(0-B), the value is binary.

For binary values, the upper 3-bits of the low hexdigit of the packed value are the high 3-bits of the binary value. The lowest bit of the last hexdigit is the sign of the value: zero for nonnegative and one for negative values.

Example:

```

10 D$=HEX(12 34 56 7C)
20 B$=HEX(2D 68 72)
30 $UNPACK (F=HEX(7004)) D$ TO X
40 $UNPACK (F=HEX(7003)) B$ TO Y

```

Results in X = 1234567 and
Y = 1234567

- Signed Binary Format (8d0y)

The Signed Binary Format is used to unpack numeric values that were packed with \$PACK format 8d0y. The maximum field length allowed is 4. The value to be unpacked is a signed binary value. Negative values are stored in 2's complement.

Example:

```
10 P$=HEX(00 2D 68 72)
20 N$=HEX(FF D2 97 8E)
30 $UNPACK (F=HEX(8004)) P$ TO X
40 $UNPACK (F=HEX(8004)) N$ TO Y
```

Results in X = 1234567 and
Y = -1234567

- Unsigned Binary Format (9d0y)

The Unsigned Binary Format is used to unpack numeric values that were packed with \$PACK format 9d0y. The maximum field length allowed is 4. The value to be unpacked is an unsigned binary value.

Example:

```
10 P$=HEX(00 2D 68 72)
20 $UNPACK (F=HEX(8004)) P$ TO X
```

Results in X = 1234567

- Compressed Alphanumeric Format (Alxx)

The \$UNPACK Compressed Alphanumeric Format is used to decompress alphanumeric data compressed with \$PACK format Alxx. Each 6-bits of the field is converted to an ASCII character. Thus, each three bytes of the field unpacks into four ASCII characters. Values hex(00) through hex(3F) are converted to the ASCII characters with values hex(20) through hex(5F). These include the uppercase characters, digits, space, and certain symbols.

If the compressed value is shorter than the receiver variable, the result is padded with trailing spaces. If the receiver is too short, the unpacked value is truncated.

Example:

```
10 P$=HEX(86 28 E4)
20 $UNPACK (F=HEX(A103)) P$ TO S$
```

Results in S\$ = "ABCDEFGH"

The Internal Form of the \$UNPACK Statement

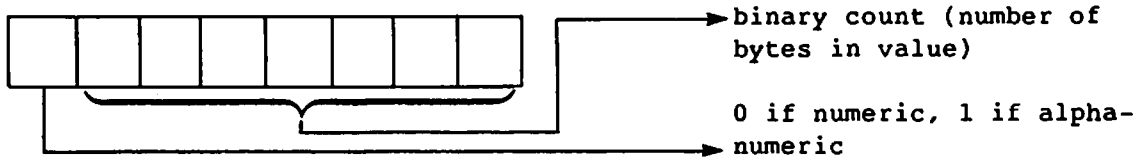
Data stored in the standard Wang 2200 disk record format can be unpacked by the internal form of \$UNPACK. Data records that have been saved on a disk platter by either the DATASAVE DC or the DATASAVE DA statement are stored in this format. Data values are sequentially read from the buffer and stored in the variables following the word TO. The unpacking terminates when the buffer is empty and the EOB (End-of-Block) character is encountered or when the entire variable list has been satisfied. An error results if a numeric value is unpacked into an alphanumeric-variable or if an alphanumeric value is unpacked into a numeric-variable.

Standard Wang 2200 Record Format (buffer format):

80	01	SOV	value	SOV	value	SOV	value	EOB
16	16							

control bytes

- The SOV (Start-of-Value) character precedes each data value in the record and indicates whether the value is numeric or alphanumeric and the length of the value.



- The EOB (End-of-Block, HEX(FD)) character indicates the end of valid data in the record.
- \$UNPACK ignores the first two control bytes.
- Data format:

Alphanumeric Values

C	C	...	C
---	---	-----	---

where:

C = any character of the alphanumeric value to be unpacked.

Numeric Values

Numeric values must be in Wang Internal Numeric Format.

se	e d	dd	dd	dd	dd	dd	dd
L	H						

where:

s = sign: 0 if mantissa +, exponent +
 1 if mantissa -, exponent +
 8 if mantissa +, exponent -
 9 if mantissa -, exponent -

e e = exponent (2 digits)
 L H

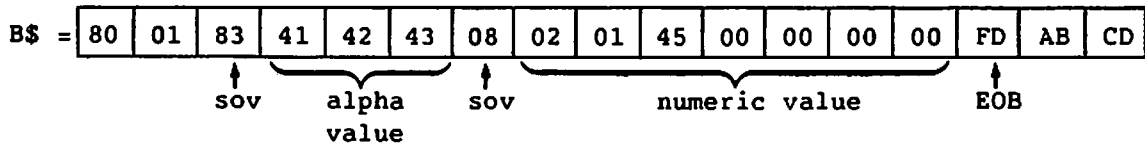
d = mantissa digit (always 13)

Leading zeros in numeric values are eliminated. All digits must be BCD.

Note: If the numeric values are invalid, the results of the conversion performed by \$UNPACK are undefined.

Example:

Suppose B\$ contains one alphanumeric value and one numeric value. The contents of B\$ are to be unpacked into the variables A\$ and X.



```
:100 $UNPACK B$ TO A$, X
:110 PRINT A$; X
:RUN
ABC 123.45
```

Examples of valid syntax:

```
$UNPACK A$ TO X
$UNPACK STR(A$,5) TO X, B$, Y
$UNPACK (F = F$) A$( ) TO X( )
$UNPACK (D = D$) A$( ) TO B$( )
$UNPACK (F = A$( )) B1$ TO A$, B$, STR(C$,3,2)
$UNPACK (D = STR(Q$,3,2)) X$( ) TO X, Y, Z(1,2)
```

DATALOAD AC (CS/386 Only)

Format

```
DATALOAD AC [file #,][record-number] alpha-variable
```

where:

```
alpha-variable = 512 bytes or larger
```

```
record-number = numeric-expression
```

The DATALOAD AC statement reads one record from a specified file of a specified disk and stores the entire 512 bytes in the designated alpha-variable. (Record = one sector = 512 bytes).

An error results if the alpha-variable is not large enough to hold at least 512 bytes. If the alpha-variable is larger than 512 bytes, the additional bytes of the array are not affected by the DATALOAD AC operation. An error will also occur if the record-number is beyond the total sectors of the file.

When a record-number is specified, the system runs Random Read mode. The statement reads one sector of the specified record which is relative to the start of the file opened. After execution, the current record-number is not affected.

If the record-number is not specified, the system is in Sequential Read mode. The statement gets the record-number from the Device Table and performs read operations. After execution, the record-number in the Device Table is automatically increased by one.

Examples of valid syntax

```
DATALOAD AC #2,(A) B$( )  
DATALOAD AC #1,(20) X$( )  
DATALOAD AC #1, A$( )
```

ALOAD AC OPEN (CS/386 Only)

Format

```
DATALOAD AC OPEN T [file #,] filename
```

The DATALOAD AC OPEN statement opens data files that have been previously stored on a MS-DOS diskette. When the statement is executed, the system finds the named file on the specified disk and sets up the starting cluster, current record-number, and file length in sectors in the Device Table. (The current record-number is set to zero, one record = one sector = 512 bytes). Any request use of the same file number in other AC statements access this file. If no file number is included, the file is assumed to be associated with the default file number (0).

An error will result if the filename cannot be found in the directory area of the specified disk or the diskette is not in MS-DOS format.

Examples of valid syntax

```
DATALOAD AC OPEN T#2, A$  
DATALOAD AC OPEN T#1, "Part.Dat"
```


DATASAVE AC (CS/386 Only)

Format

```
DATASAVE AC [file#,) [(record-number)] (literal-string)
                                         (alpha-variable)
```

where:

```
record-number = numeric-expression
```

The DATASAVE AC statement writes one record to the disk. The alpha-variable or literal-string contains the data to be written. (Record = one sector = 512 bytes). If the data is longer than 512 bytes, the first 512 bytes are written. If the data is shorter than 512 bytes, the remainder of the sector is filled with zeros.

If record-number is specified, the system runs Random Write mode. The data is written into the record-number which is relative to the start of the opened file. After execution, no information in the Device Table will be altered.

When record-number is not specified, the system is in Sequential Write mode. The statement gets the record-number from the Device Table and performs write operations. The record-number in the Device Table automatically increases by one after execution.

Examples of valid syntax

```
DATASAVE AC #2,(1) AS( )
DATASAVE AC #1, AS( )
```

DATASAVE AC OPEN (CS/386 Only)

Format

```
DATASAVE AC OPEN T [file #] (old-filename)
                    ((size in sector) new-filename)
```

where:

```
Size = numeric-expression
```

The DATASAVE AC OPEN statement creates a new DOS file or rewrites an existing file.

If creating a new file, space is reserved in the disk and a file entry is made in the directory. (Space = sector number specified in size). The disk on which the file is stored, along with the file starting cluster, record-number (initially set to zero), and total sectors are entered in the Device Table. An error will occur if there is not enough space.

When rewriting an old file, the operations are the same as the DATALOAD AC OPEN statement.

Examples of valid syntax

```
DATASAVE AC OPEN T#2, (10) "Data1.Dat"
DATASAVE AC OPEN T#3, "Data1.Dat"
```

DATASAVE AC CLOSE (CS/386 Only)

Format

DATASAVE AC CLOSE file#

The DATASAVE AC CLOSE statement closes an individual data file if it is no longer used in the current or sequential program.

Example of valid syntax

DATASAVE AC CLOSE #1

DATASAVE AC END (CS/386 Only)

Format

```
DATASAVE AC [file#,) END
```

After execution of the DATASAVE AC OPEN statement, the disk space is allocated for use. Completing the DATASAVE AC operations, you may find unused space. The DATASAVE AC END statement enables you to update the file length and return the unused disk space.

The DATASAVE AC END statement runs in sequential write mode. The statement takes the record-number from the Device Table as the total file sector number.

Restrictions

File# must be 0-15

You can shorten file length only; i.e., no expanded file length

Example of valid syntax

```
DATASAVE AC #1, END
```

DBACKSPACE

Format:

DBACKSPACE [file#,]	BEG
	numeric-expression [S]

The DBACKSPACE statement backspaces over logical records or sectors within a cataloged disk file. If a value is specified with a numeric expression and S is not specified, the system backspaces over the number of logical records equal to the value of the numeric-expression, and the Current Sector Address of the file in the Device Table is updated to the starting sector of the new logical record. For example, if numeric-expression = 1, the Current Sector Address is set equal to the starting address of the previous logical record.

If the BEG parameter is used, the Current Sector Address is set equal to the Starting Sector Address of the file (i.e., the starting address of the first logical record in the file).

If the S parameter is used, the value of the expression equals the total number of sectors to backspace. The Current Sector Address of the file in the Device Table is decreased by the number of sectors specified. If the amount specified is too large, the Current Sector Address is set to the starting Sector Address of the file. The S parameter is particularly useful in files where all the logical records are of the same length (i.e., have the same number of sectors per logical record). Backspacing with the S parameter is much faster than backspacing over logical records in a file since the system merely decreases the Current Sector Address in the Device Table by the specified number of sectors and no disk accesses are required. However, you must be certain that you know exactly how many sectors are in each logical record.

Examples of valid syntax:

```
DBACKSPACE BEG
DBACKSPACE 2*X
DBACKSPACE #2, 5S
DBACKSPACE #1, BEG
DBACKSPACE #A, 10
```



INDEX

A

ABS function, 4-7, 4-9, 4-12,
6-4, 6-6, 9-9
access modes, 10-1, 10-7
ADD operator, 5-9 to 5-10, 6-1,
6-6 to 6-8
addition, 4-4, 6-1, 6-7 to 6-9
\$ALERT statement, 8-2, 8-4, 8-5
ALL function, 5-9 to 5-10, 5-12,
5-15, 6-3 to 6-8, 6-13
alphanumeric
 argument, 6-6
 array, 5-2
 character String, 4-2, 5-1
 expression, 1-2, 5-8, 5-10
 7-20
 literal string, 5-4 to 5-5,
 5-17, 5-28, 6-1
 operator, 5-9
 string manipulation, 5-9
 variable, 1-3, 1-5, 5-1 to 5-2,
 5-4, 5-10 to 5-11, 5-18, 5-20,
 5-28, 6-9 to 6-12, 10-26
alternate character set, 3-1, 3-13
3-14 to 3-15
AND operator, 5-6, 5-8, 5-9 to 5-12
5-14 to 5-16
ARCCOS function, 4-8, 4-13
ARCSIN function, 4-8, 4-13
ARCTAN function, 4-8, 4-13
argument, 3-16, 4-9 to 4-13, 5-9,
5-13, 5-18, 5-20, 5-23, 5-27,
5-28, 6-6, 9-5, 9-9,
arithmetic operators, 4-1, 4-4,
4-5 to 4-6, 5-1, 6-4, 6-6
array

 alphanumeric, 5-7
 dimensioning of, 4-4, 10-14, 13-2
 locator, 14-2
 numeric, 4-3, 4-7, 4-9, 10-14,
 10-17
 redimensioning of, 13-2, 13-3
 variables, 5-2, 5-11
assignment statement, 1-5, 4-4,
5-5 to 5-11, 5-13 to 5-14,
5-25, 6-7, 6-9 to 6-10, 6-12,
7-27, 9-5, 10-28
ATN function, 4-8

B

background processing, 16-12
BASIC statements, 1-3 to 1-5, 4-2,
4-4, 11-5
BIN function, 4-14, 5-9, 5-13, 5-27
binary, 4-1, 4-14, 5-1, 5-9, 5-10,
5-13, 5-15, 5-27
 conversion, 6-2
 addition, 6-1, 6-7, 6-9
 subtraction, 6-1, 6-10, 6-12
 operators, 6-1, 6-6
blanks
 initial value of branching,
 10-26
 trailing, 5-3
BOOL operator, 5-14 to 5-15
bootstrapping, 16-10
box graphics, 3-1, 3-15 to 3-16
branching, 1-4 to 1-6, 7-19, 8-9,
9-4, 10-28, 10-30,
\$BREAK statement, 16-36
broadcast message, 16-11

INDEX (continued)

C

character display attributes, 3-1,
 3-3 to 3-4, 3-7,
 blinking, 3-3 to 3-4
 bright, 3-3
 reverse video, 3-3
 character format, 5-29
 character graphics, 3-1, 3-13 to
 3-16
 character mode, 3-15
 character sets, 3-1, 3-13, 5-29
 alternate, 3-13 to 3-15
 normal, 3-13 to 3-14
 selection of, 3-13 to 3-14
 character string, 4-2, 4-14, 5-1,
 5-2, 5-3, 5-4, 5-7 to 5-8,
 5-10, 5-17, 5-21, 5-22, 5-24,
 5-26, 6-1, 10-2, 10-7, 10-15,
 CLEAR command, 4-11, 4-13, 7-17,
 8-3 to 8-5, 8-11, 9-3, 9-9,
 10-1, 10-4, 10-10, 10-27,
 10-28
 CLEAR N, 10-5, 10-6, 10-8
 CLEAR P, 2-6, 10-4, 10-5, 10-6,
 10-8
 CLEAR V, 10-4, 10-5, 10-6, 10-8
 \$CLOSE statement, 16-38
 code conversion, 5-13, 5-27, 6-2
 to 6-6,
 COM CLEAR statement, 11-9
 COM statement, 1-3, 4-3, 5-2,
 10-14, 10-26, 11-6
 communication
 asynchronous, 7-18, 7-26
 concatenation operator (&), 5-6
 5-9
 condition code, 4-14
 console operations, 7-4, 7-6, 7-8,
 7-10 to 7-13, 7-16 to 7-18,
 10-4, 10-10, 10-28
 constants
 numeric, 4-1 to 4-2, 4-4
 CONTINUE command, 2-10, 10-1,
 10-2, 10-6, 10-7, 10-8, 10-25,
 10-27,

control codes, 3-1, 3-2, 3-3,
 3-12, 5-5, 5-17, 5-20, 10-3,
 10-4, 10-25
 CONVERT statement, 5-21, 10-13,
 11-11
 COPY statement, 12-21
 COS function, 4-8, 4-13

D

DAC operator, 5-9, 6-1, 6-6, 6-9
 data pointer, 10-26
 data types
 alphanumeric, 1-3, 4-1, 5-2, 6-1
 numeric, 4-1, 5-1, 14-2
 DATA statement, 1-3 to 1-4, 11-15,
 5-27
 DATALOAD BA, 12-22
 DATALOAD BM, 12-23
 DATALOAD DA, 12-24
 DATALOAD DC, 12-11, 12-12, 12-26,
 12-27
 DATALOAD AC, 11-36a
 DATASAVE BA, 12-28
 DATASAVE BM, 12-29
 DATASAVE DA, 12-30
 DATASAVE DC, 12-7 to 12-8, 12-32,
 12-34, 12-35,
 DATASAVE AC, 11-36c
 DATE function, 5-16, 5-27
 debugging, 1-1, 1-2, 1-4, 2-1,
 2-8, 2-9, 4-10, 7-3, 8-3,
 10-27, 10-28
 DEFFN @PART statement, 16-39
 DEFFN statement, 1-4, 11-17
 DEFFN' statement, 1-4, 2-11, 9-4,
 10-7, 10-22, 11-20
 DELETE statement, 2-5
 deleting
 characters, 2-2
 lines, 2-6, 4-11
 device address, 7-1, 7-6 to 7-27,
 8-6, 8-8
 Device Table, 2-11, 7-1, 7-5, 7-7
 to 7-27, 10-2, 10-4
 device type, 7-6 to 7-27
 devices
 control of, 5-17, 7-1

INDEX (continued)

digit specifiers, 7-25
DIM statement, 4-3 to 4-4, 5-2,
5-5, 5-7, 5-18, 5-22 to 5-23,
6-2 to 6-13
\$DISCONNECT statement, 16-41
division, 4-4 to 4-5, 4-10, 4-13,
4-14,
by Zero, 9-5, 9-9
of character space, 3-14
DO group, 9-6 to 9-7, 11-29
dollar sign (\$), 5-1
DSC operator, 5-9, 6-1, 6-5, 6-6,
6-10

E

edit mode, 1-2, 1-4, 2-3
END statement, 1-6, 10-11 to
10-14, 10-21, 10-30, 11-30,
4-14, 5-16 to 5-17, 5-26, 9-1
to 9-3
ERR function, 9-3
error conditions, 4-14, 5-16 to
5-17, 5-26, 9-1 to 9-3
error detection, 9-1, 10-26
error messages, 1-4 to 1-6, 2-8 to
2-9, 4-1, 4-6, 4-14, 7-2 to
7-3, 7-12, 9-1 to 9-2, 9-5 to
9-6, 9-8, 10-23
error recoverability, 9-2
ERROR statement, 9-1 to 9-9
evaluation
of numeric expressions, 4-1,
4-4 to 4-6
executable statements, 1-2, 1-3,
4-6
exponentiation, 4-1, 4-2, 4-5, 4-6
expressions
alphanumeric, 1-2, 5-8, 5-10,
5-14, 5-16, 6-7, 6-9 to 6-11,
7-20
numeric, 4-1 to 4-5, 4-9, 5-8,
7-1, 7-20, 7-23 to 7-24, 8-5
evaluation of, 5-13

F

file
catalog mode, 12-1, 12-3
number, 7-13 to 7-16, 7-21, 7-25
to 7-26
status, 7-15
FIX function, 4-7 to 4-9, 4-12
floating point format, 4-1
FOR statement, 1-5, 4-11, 6-2 to
6-4, 10-25 to 10-29
foreground processing, 16-12
\$FORMAT statement, 11-35, 12-39
function
alphanumeric, 5-1, 5-13, 5-16,
7-27, 9-5
defined, 1-2, 10-7
keys, 10-7
numeric, 4-4, 4-6 to 4-8, 4-13
to 4-14, 5-9, 5-18, 5-20,
5-23, 5-26 to 5-28, 9-3

G

\$GIO statement, 7-6, 7-13, 15-1,
15-8
global
partition, 6-14, 16-24
subroutines, 16-18
text, 16-15, 16-19
variable, 8-5, 16-17, 16-20,
16-22
GOSUB statement, 1-4, 5-22, 6-4,
6-6, 8-5, 8-6, 9-4, 10-9,
10-21, 10-23,
GOSUB' statement, 1-4, 9-3, 9-6,
10-22, 11-38
GOTO statement, 1-4 to 1-5, 4-4,
5-21, 6-4, 6-6, 10-8, 10-21,
10-23, 10-26, 10-28, 11-40
graphics
box, 3-1, 3-15 to 3-16
character, 3-1, 3-13, 3-15

INDEX (continued)

H

HALT function, 1-6, 2-10, 7-6,
7-12, 10-1 to 10-2, 10-6,
10-8, 10-9 to 10-10, 10-25,
10-27
hex codes, 3-1, 3-2, 5-1, 5-5
attributes, 3-3
used to turn off attributes,
3-4, 3-7
hex control codes, 3-12, 3-13
HEX function, 3-1, 5-5, 5-17
hexadecimal
literal strings, 3-1, 5-5 to
5-6, 5-17
representation, 3-1, 5-10,
5-14 to 5-15, 5-22, 6-9 to
6-10, 7-6, 7-25
HEXPACK statement, 11-41
HEXUNPACK statement, 11-44

I

IF statement, 2-7, 5-11, 5-17,
5-19, 5-21, 5-23, 5-25, 5-27,
6-2 to 6-7, 9-3 to 9-4, 10-21,
11-45
\$IF ON/OFF statement, 15-5
IF ... THEN statement, 11-45
Image (%) statement, 2-2, 9-6,
11-48
Immediate mode, 1-2 to 1-6, 2-1 to
2-7, 2-9 to 2-10, 7-3, 7-6,
7-10, 7-11 to 7-13, 8-8, 10-1,
10-6, 10-8, 10-27,
\$INIT command, 16-45
INPUT statement, 1-4, 2-6, 2-8,
5-21, 7-4 to 7-6, 7-8, 7-11,
7-12, 7-17 to 7-18, 7-22,
7-25, 9-7, 10-4, 10-12, 10-13,
11-49
INT function, 4-7 to 4-10, 4-12
internal
decimal format, 5-13
interrupt
control statements, 8-1 to 8-4
Interrupt Table, 8-6, 8-11, 10-2
I/O

bus, 15-2
operations, 7-1, 7-6 to 7-8,
7-10, 7-16, 7-18, 8-2, 8-5,
16-62
statements, 15-4

K

keyboard, 3-1, 4-1, 5-1, 5-5,
5-17, 7-6 to 7-8, 8-2, 10-1,
10-3, 10-7, 10-25
Keys, 2-3 to 2-5, 2-7 to 2-8, 3-13
KEY function, 10-7
KEYIN statement, 7-6, 7-11 to
7-12, 7-18, 8-2, 10-4, 10-7,
10-21, 11-52,
Keyword, 1-6, 3-6

L

label, 10-11, 11-26
LEN function, 4-14, 5-3, 5-8 to
5-9, 5-18 to 5-19, 6-2 to 6-4
LET statement, 4-4, 10-28, 11-54
LGT function, 4-7
line
blank, 10-10
editing, 2-1 to 2-3
feed, 2-1, 3-2, 3-9
length, 2-2
multistatement, 1-5, 10-8
renumbering, 2-6 to 2-7
replacement, 2-6
width, 7-3 to 7-5, 7-10 to 7-13,
7-17, 7-21,
line numbers, 1-3 to 1-6, 2-7 to
2-11, 10-1 to 10-6, 10-8,
10-11 to 10-12, 10-20, 10-22
to 10-23, 10-26 to 10-30
LINPUT statement, 1-4, 3-11, 3-15,
5-22, 5-25, 7-6, 7-11 to 7-12,
7-18, 10-4, 11-55
LIST command, 2-8, 2-10, 7-1, 7-5,
7-7, 7-11, 7-21, 10-2, 10-4,
10-10, 10-12, 10-22,
LIST D, 2-10, 3-11, 10-11 to
10-13 10-13

INDEX (continued)

LIST DC, 12-42
 LIST DT, 2-11, 7-1, 7-9, 7-20 to
 7-22, 10-2,
 LIST I, 8-3 to 8-4, 8-6 to 8-7,
 10-2
 LIST T, 2-11, 10-2, 10-15 to
 10-16
 LIST V, 10-17
 literal strings, 2-2, 3-1, 5-1,
 5-4, 5-6, 5-10, 5-16, 5-22,
 5-26 to 5-28, 8-6, 10-10,
 10-14 to 10-16, 5-22, 5-26,
 alphanumeric, 5-4 to 5-5, 5-17,
 6-1
 hexadecimal, 5-5 to 5-6, 5-17
 LOAD command, 1-6, 7-11, 8-5, 9-3
 to 9-4, 10-2, 10-11, 12-4,
 12-45 to 12-46,
 LOAD DA, 12-48 to 12-49
 LOAD RUN command, 7-4 to 7-5,
 7-10, 7-17, 9-9, 10-2, 10-10,
 12-6, 10-27, 12-51,
 LOG function, 1-5, 2-6, 4-7, 9-9
 logical operators, 5-1, 5-11,
 5-15
 loops, 10-25 to 10-28
 nested, 4-13

M

Master Device Table, 7-7, 7-21
 master initialization, 4-6, 4-11,
 4-13, 7-2 to 7-5, 7-8, 7-10,
 7-16 to 7-17, 9-9, 10-10
 matrix Statements, 13-3
 memory
 allocation, 16-5
 message, 1-6, 2-8 to 2-9, 4-6,
 7-11 to 7-12, 10-4, 10-25,
 modes
 box graphic, 3-16
 character, 3-15
 edit, 2-3
 immediate, 2-1 to 2-7, 2-9,
 2-10, 8-8, 9-6, 10-1
 operator's, 1-3
 programmer's, 1-4, 1-6
 MAT +, 13-4

MAT CON, 13-5
 MAT COPY, 11-58
 MAT =, 13-6
 MAT IDN, 13-7
 MAT INPUT, 13-8
 MAT INV, 13-10
 MAT *, 13-13
 MAT MERGE, 14-4
 MAT MOVE, 11-60, 14-13
 MAT PRINT, 13-14
 MAT READ, 13-15
 MAT REDIM, 13-17
 MAT ()*, 13-19
 MAT -, 13-20
 MAT SEARCH, 11-64
 MAT SORT, 14-18
 MAT TRN, 13-21
 MAT ZER, 13-22
 MAX function, 4-7, 4-9
 MIN function, 4-9
 MOD function, 4-6 to 4-7, 4-10,
 7-2,
 MOVE statement, 12-52 to 12-54
 \$MSG statement, 16-49
 multiplication, 4-4 to 4-5

N

names
 array, 4-3, 5-2, 5-7
 file, 7-15
 variable, 4-3, 5-1, 5-4, 10-28
 nested loops, 4-13
 NEXT statement, 1-5 to 1-6, 4-11,
 6-2 to 6-3, 10-25 to 10-29
 NUM function, 4-14, 5-9, 5-20
 numeric
 constants, 4-2
 expressions, 1-2 to 1-3, 4-4,
 5-8, 7-1, 7-20, 7-23 to 7-25,
 8-5
 functions, 4-6, 4-13, 5-9, 5-18,
 5-23, 5-27 to 5-28, 9-3
 values, 4-1, 5-23, 5-27, 6-2
 variables, 4-3, 5-1, 5-4, 7-19,
 7-25, 7-27, 8-8, 10-26,

INDEX (continued)

O

ON/GOSUB statement, 1-4, 7-19, 11-70
 ON/GOTO statement, 1-4, 7-19, 1026, 11-70
 ON/SELECT statement, 7-19 to 7-20, 7-23 to 7-24
 \$OPEN statement, 16-50
 operators, 1-2 to 1-3
 alphanumeric, 6-1
 arithmetic, 4-1, 4-4 to 4-6, 5-1, 6-4, 6-6
 binary, 6-1
 concatenation, 5-6, 5-9
 logical, 5-1, 5-10 to 5-11, 5-14
 relational, 5-11, 5-14, 5-22, 8-4
 OR operator, 5-9, 5-11 to 5-12, 5-15
 output
 devices, 7-1, 7-3 to 7-5, 7-11 to 7-13, 7-21, 7-26
 formatting, 7-3

P

parameters
 selecting, 7-11 to 7-16
 partitions, 4-11, 4-13 to 4-14, 7-9, 7-21, 8-2, 8-5, 8-8
 PACK statement, 6-4 to 6-6, 6-10, 11-72
 \$PACK statement, 11-74
 parenthesis
 specifying Arrays, 4-3, 5-7
 specifying Order of, 5-9
 evaluation, 4-5
 #PART function
 partition
 global, 16-22, 16-24
 pause mode, 7-3 to 7-4, 10-4, 10-25
 peripheral
 allocation, 16-9
 POS function, 4-14, 5-9, 5-22 to 5-23
 PRINT AT statement, 3-2, 3-12, 3-16, 10-15, 11-93

PRINT BOX statement, 3-1, 3-15 to 3-16, 11-95
 PRINT HEX statement, 3-2, 3-5 to 3-11, 11-97
 PRINT statement, 1-4 to 1-5, 3-6, 3-11, 4-2, 4-4 to 4-6, 4-11, 4-14, 5-4, 7-4, 7-9 to 7-10, 7-12, 10-4, 11-84
 Printer Device Table, 7-21
 printer driver, 7-26
 printers, 7-3 to 7-4, 7-7, 7-12 to 7-13, 7-23, 8-1,
 PRINTUSING statement, 3-2, 7-4, 7-6, 7-10, 7-12, 10-4, 10-23, 11-99, 11-107
 program execution, 1-3, 2-9 to 2-10, 4-2 to 4-3, 4-6, 4-13, 7-2, 7-19, 9-1 to 9-2, 9-6 to 9-8, 10-1 to 10-3, 10-6 to 10-8, 10-25, 10-27 to 10-28
 program lines, 1-4 to 1-6, 2-1 to 2-2, 2-5 to 2-7, 2-11
 program mode, 1-3
 program statements, 1-5, 5-6
 program termination, 7-2, 9-6
 prompting input, 9-1
 \$PSTAT statement, 16-52

Q

question mark, 10-22
 quotes, 5-4, 5-17

R

RAMDISK, 12-18 to 12-19
 READ statement, 1-6, 5-25, 9-7, 10-24, 10-26, 11-109
 \$RELEASE statement, 16-57
 REM statement, 1-3, 3-11, 6-2 to 6-6, 8-5, 8-11, 10-11 to 10-15, 10-21, 10-24, 10-30, 11-110, 12-55
 RENAME statement, 10-2, 12-4
 RENUMBER command, 10-3, 10-6, 10-8, 10-23 to 10-24
 RESAVE statement, 12-56, 12-4

INDEX (continued)

RESET, 1-6, 3-12, 3-14, 7-4 to
 7-5, 7-16 to 7-17, 10-3, 10-6,
 10-8, 10-25, 10-27 to 10-28
 resolution phase, 1-6, 9-1
 RESTORE statement, 3-8, 5-27,
 11-112
 RETURN CLEAR statement, 11-116
 RETURN statement, 1-4, 2-5, 6-2 to
 6-6, 8-2 to 8-3, 10-11, 10-21,
 10-24, 10-26, 10-28, 10-30,
 11-114
 RND function, 4-7, 4-10 to 4-11
 ROTATE statement, 11-117
 ROUND function, 4-6 to 4-7, 4-11
 to 4-12
 RUN command, 1-6, 5-3 to 5-6,
 5-18, 5-20, 5-21, 5-23 to
 5-25, 6-7 to 6-13, 7-4, 7-11,
 8-5, 10-3, 10-7 to 10-8, 10-26
 to 10-27

S

SAVE statement, 5-22, 10-3, 12-57,
 12-60, 12-3
 scalar variable
 alphanumeric, 5-2, 5-7, 10-17
 numeric, 4-3, 10-17, 10-19
 SCRATCH statement, 12-61 to 12-62
 SELECT ERROR statement, 1-6, 4-7,
 7-2, 9-2 to 9-3, 9-6 to 9-9
 SELECT NO ROUND, 4-6, 7-2
 SELECT ON CLEAR statement, 8-11
 SELECT ON/OFF statement, 8-1 to
 8-11
 SELECT P, 2-10
 SELECT @PART, 16-59
 SELECT ROUND, 4-6
 SELECT statement, 7-1 to 7-3, 7-6
 to 7-7, 7-10, 7-14 to 7-16,
 7-19 to 7-20, 7-25 to 7-26,
 8-1, 10-10, 10-16
 semicolon, 5-4, 7-19 to 7-20, 7-23
 SGN function, 4-7, 4-9, 4-12
 SIN function, 4-8, 4-13
 sort
 numeric data, 14-2
 statements, 14-3
 SPACE function, 4-13 to 4-14

SPACEK function, 4-13, 4-14
 SPACE S & SK, 4-13, 4-14
 SQR function, 4-4, 4-6 to 4-7, 7-2
 STOP statement, 1-6, 10-1, 10-3,
 10-6, 10-8 to 10-9, 10-24,
 10-27, 11-119
 STR function, 6-2 to 6-9, 6-12 to
 6-13, 5-3, 5-24
 strobe
 address, 15-3
 input, 15-4
 output, 15-3
 SUB operator, 5-9, 6-1, 6-12 to
 6-13
 subroutine, 7-16, 9-6, 10-1 to
 10-2, 10-5, 10-7, 10-22, 10-25
 to 10-26
 interrupt, 8-1 to 8-9
 subscript, 4-3 to 4-4, 5-27
 substring, 5-3 to 5-4, 5-11, 5-24
 to 5-25
 subtraction, 4-5
 binary, 6-1, 6-10, 6-12
 decimal, 6-1, 6-10
 syntax
 errors, 9-1 to 9-2, 10-17,
 10-19, 10-21, 10-24, 10-26,
 10-27, 10-30
 valid examples of, 4-5, 5-10 to
 5-27, 6-8 to 6-13, 7-22, 7-24,
 7-27, 8-5, 8-7, 8-10 to 8-11,
 9-4 to 9-7, 9-9, 10-5 to 10-6,
 10-13 to 10-16
 system commands, 1-2, 1-4, 10-3

T

TAN function, 4-8, 4-13
 text entry mode, 5-26, 5-16
 TRACE command, 2-10, 7-3, 7-6,
 7-12, 10-3, 10-8 to 10-9,
 10-28, 10-30
 \$TRAN statement, 11-120
 truncation, 4-6, 4-9, 4-11

U

UNPACK statement, 6-6, 11-124
 \$UNPACK statement, 11-125

INDEX (continued)

V

VAL function, 4-14, 5-9, 5-13,
5-27, 6-4

variable, 2-9 to 2-11, 4-1 to 4-4,
4-9, 5-1

VER function, 4-14, 5-9, 5-28, 5-29

VERIFY statement, 5-22, 12-64

X

XOR operator, 5-10, 5-11 to 5-12,
5-15

Z

zoned format, 3-12



Title MULTIUSER BASIC-2 LANGUAGE REFERENCE MANUAL

Help Us Help You . . .

We've worked hard to make this document useful, readable, and technically accurate. Did we succeed? Only you can tell us! Your comments and suggestions will help us improve our technical communications. Please take a few minutes to let us know how you feel.

How did you receive this publication?

Support or Sales Rep

Wang Supplies Division

From another user

Enclosed with equipment

Don't know

Other _____

How did you use this Publication?

Introduction to the subject

Classroom text (student)

Classroom text (teacher)

Self-study text

Aid to advanced knowledge

Guide to operating instructions

As a reference manual

Other _____

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR	VERY POOR
Technical Accuracy — Does the system work the way the manual says it does?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Readability — Is the manual easy to read and understand?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Clarity — Are the instructions easy to follow?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Examples — Were they helpful, realistic? Were there enough of them?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization — Was it logical? Was it easy to find what you needed to know?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Illustrations — Were they clear and useful?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Physical Attractiveness — What did you think of the printing, binding, etc?	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Were there any terms or concepts that were not defined properly? Y N If so, what were they? _____

After reading this document do you feel that you will be able to operate the equipment/software? Yes No Yes, with practice

What errors or faults did you find in the manual? (Please include page numbers) _____

Do you have any other comments or suggestions? _____

Name _____ Street _____

Title _____ City _____

Dept/Mail Stop _____ State/Country _____

Company _____ Zip Code _____ Telephone _____

Thank you for your help.

WANG

Fold



WANG

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

**Wang Laboratories, Inc.
Technical Publications Dept.
M/S 012-260
One Industrial Avenue
Lowell, Massachusetts 01851-9971**

NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES



Fold



Cut along dotted line

Order Form for Wang Manuals and Documentation

① Customer Number (If Known) _____				
② Bill To: _____			Ship To: _____	
_____			_____	
_____			_____	
_____			_____	
③ Customer Contact: _____		④ Date _____	Purchase Order Number _____	
() () _____				
Phone _____		Name _____		
⑤ Taxable Yes <input type="checkbox"/> No <input type="checkbox"/>		⑥ Tax Exempt Number _____		⑦ Credit This Order to _____
		A Wang Salesperson _____		_____
		Please Complete _____		Salesperson's Name _____
				Employee No. _____
				RDB No. _____
⑧ Document Number	Description	Quantity	⑨ Unit Price	Total Price
⑩ _____ Authorized Signature _____ Date _____ <input type="checkbox"/> Check this box if you would like a free copy of WangDirect Software & Literature Catalog (711-0888A)			Sub Total	
			Less Any Applicable Discount	
			Sub Total	
			Local/State Tax	
			Total Amount	

Ordering Instructions

1. If you have purchased supplies from Wang before, and know your Customer Number, please write it here.
2. Provide appropriate Billing Address and Shipping Address.
3. Please provide a phone number and name, should it be necessary for WANG to contact you about your order.
4. Your purchase order number and date.
5. Show whether order is taxable or not.
6. If tax exempt, please provide your exemption number.
7. If you wish credit for this order to be given to a WANG salesperson, please complete.
8. Show part numbers, description and quantity for each product ordered.
9. Pricing extensions and totaling can be completed at your option; Wang will refigure these prices and add freight on your invoice.
10. Signature of authorized buyer and date.

Wang Terms and Conditions

1. **TAXES** — Prices are exclusive of all sales, use, and like taxes.
2. **DELIVERY** — Delivery will be F.O.B. Wang's plant. Customer will be billed for freight charges; and unless customer specifies otherwise, all shipments will go best way surface as determined by Wang. Wang shall not assume any liability in connection with the shipment nor shall the carrier be construed to be an agent of Wang. If the customer requests that Wang arrange for insurance the customer will be billed for the insurance charges.
3. **PAYMENT** — Terms are net 30 days from date of invoice. Unless otherwise stated by customer, partial shipments will generate partial invoices.
4. **PRICES** — The prices shown are subject to change without notice. Individual document prices may be found in the WangDirect Software & Literature Catalog (711-0888A)
5. **LIMITATION OF LIABILITY** — In no event shall Wang be liable for loss of data or for special, incidental or consequential damages in connection with or arising out of the use of or information contained in any manuals or documentation furnished hereunder.

WANG



Cut along dotted line

Fold



NO POSTAGE
NECESSARY IF
MAILED IN THE
UNITED STATES

WANG

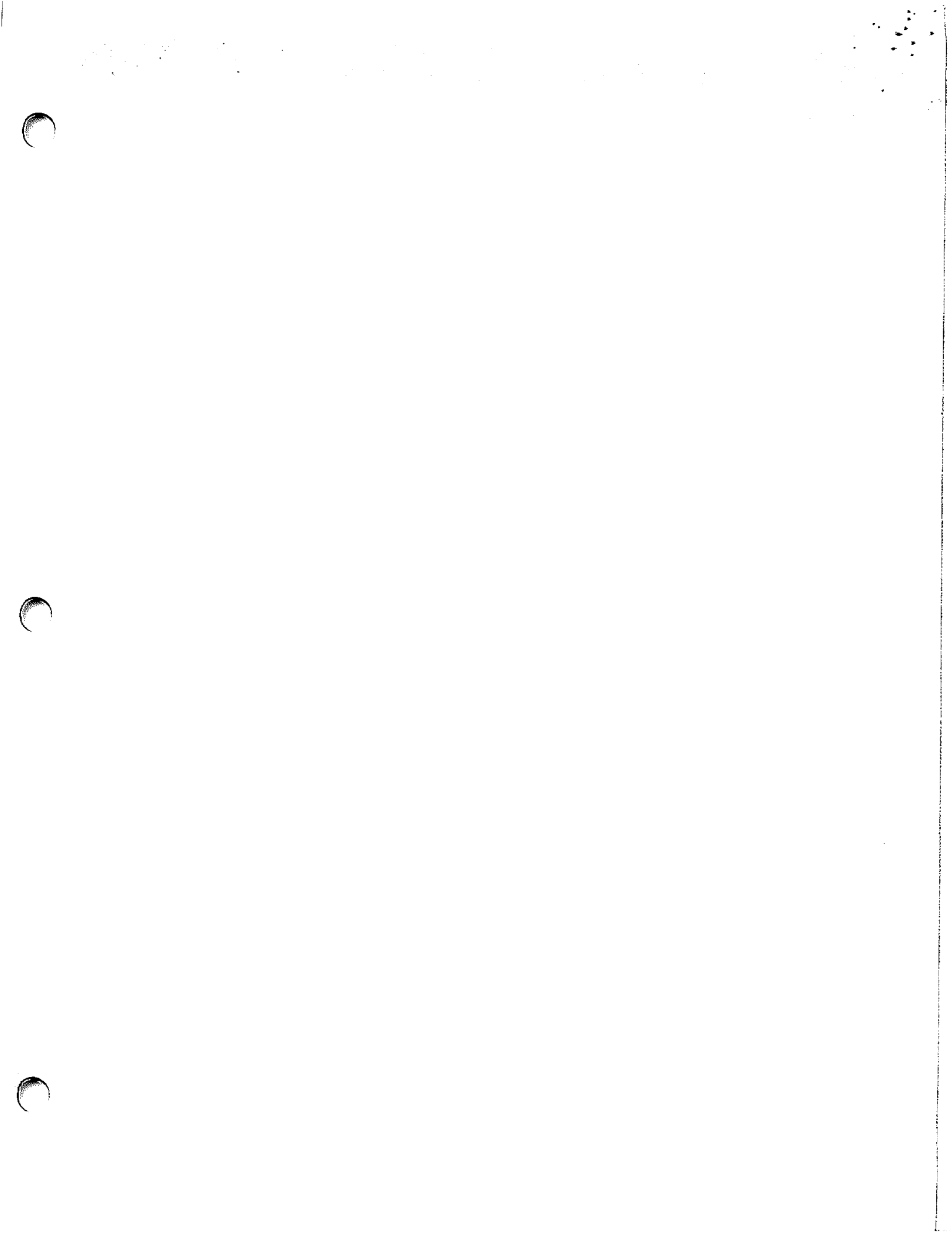
BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 16 LOWELL, MA
POSTAGE WILL BE PAID BY ADDRESSEE

**WangDirect
Wang Laboratories, Inc.
M/S 017-110
800 Chelmsford Street
Lowell, Massachusetts 01851-9972**



Fold





WANG

ONE INDUSTRIAL AVENUE, LOWELL, MA 01851
TEL. (508) 459-5000, TELEX 172108