WANG LABORATORIES, INC., ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL: 508/459-5000, TELEX 172108

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	Insert
Chapter 4 Pages 4-14	Chapter 4 Pages 4-14, 4-15
Chapter 5 Pages 5-13, 5-27	Chapter 5 Pages 5-13, 5-27
Chapter 7 Pages 7-25, 7-26	Chapter 7 Pages 7-25, 7-26
Chapter 11 Pages 11-77 through 11-84 11-129 through 11-136	Chapter 11 Pages 11-77 through 11-84 11-129 through 11-136
Chapter 12 Pages - None	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

CHAPTER 2 EDITING AND DEBUGGING FEATURES

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

CHAPTER 7 THE SELECT STATEMENT

CHAPTER 8 PROGRAMMABLE INTERRUPTS

CONTENTS (continued)

CONTENTS (continued)

CONTENTS (continued)

CHAPTER 12 DISK I/O STATEMENTS

CONTENTS (continued)

CONTENTS (continued)

FIGURES

TABLES

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
Note: * CS/386 ONLY		

BIN Function

Format:

```
BIN( expression [,lenght] )
```

where:

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

      If lenght= 1, 0 <= value-of-expression < 256
      If lenght= 2, 0 <= value-of-expression < 65,536
      If lenght= 3, 0 <= value-of-expression < 16,777,216
      If lenght= 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:

        ⎧alpha-variable ⎫
   VAL( ⎨               ⎬ [,length] )
        ⎩literal-string ⎭

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:

          ⎧alpha-variable⎫
     VER( ⎨              ⎬  , format-specification )
          ⎩literal-string⎭

where:

                                   ⎧alpha-variable⎫
     format-specification =        ⎨              ⎬
                                   ⎩literal-string⎭
```

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:

                            { 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)]        }
    select-parameter =      {                                       }
                            { 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 sitution, 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
A1xx	compressed alphanumeric format

where:
xx = field width in binary (xx > 0)
y = field width in binary (0 < y < =4)
d = implied decimal position in binary

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)

s	d	d	...	d	.	d	...	d

s	d	.	d	d	d	d	d	d	d	d	E	±	d	d

sign mantissa exponent

where:

s = sign (space if value $\geq$ 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)

s	d	d	...	d

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.

```
  Statements                        Results

  :10 F$ = HEX(100A)
  :20 $PACK (F = F$) B$ FROM X      B$ = "12.345"

  :10 F$ = HEX(240A)                B$ = "+000123450"
  :20 $PACK (F = F$) B$ FROM X      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)                Results in an error because the
  :20 $PACK (F = F$) B$ FROM X      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 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

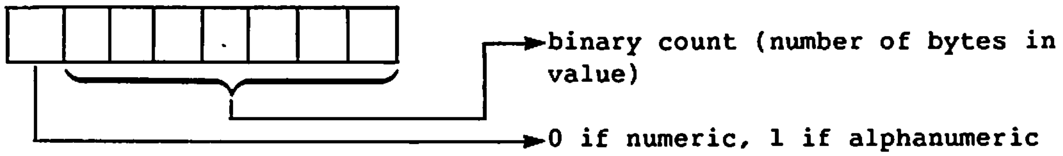
The internal form of $PACK stores data in the standard Wang 2200 disk record format. The values of the variables and arrays in the variable list are sequentially packed into the buffer. The packing terminates when all values have been packed.

Standard Wang 2200 Record Format (buffer format)

80_{16}	01_{16}	SOV	value	SOV	value	SOV	value	EOB

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.


binary count (number of bytes in value)
0 if numeric, 1 if alphanumeric

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

Data format:

Alphanumeric Values

C	C	...	C

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.

Numeric Values

Numeric values are stored in Wang Internal Numeric Format.

s	e_L	e_H	d	d	d	d	d	d	d	d	d	d	d	d	d	(8 bytes)

where:

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

$e_L e_H$ = exponent (2 digits)

d = mantissa digit (always 13)

Numeric values are normalized (i.e., leading zeros are eliminated). All digits must be BCD.

Example:

The following routine packs the values of A$ and X into a buffer B$ in internal format:

```
:10 DIM B$20, A$5
:20 A$ = "ABC" :X = 123.45
:30 $PACK B$ FROM A$, X
```

The resulting buffer appears as follows:

B$ =	80	01	85	41	42	43	20	20	08	02	01	23	45	00	00	00	00	FD	20	20
			SOV	value of A$					SOV	value of X								EOB		

Examples of valid syntax:

```
$PACK A$ FROM X
$PACK STR(B$,3,8) FROM X, A$
$PACK (F = X$) B$( ) FROM X( )
$PACK (D = D$) B1$( ) FROM B$( ), X, Y, A$
```

PRINT

Format:

$$\text{PRINT [print-element]} \left[\begin{Bmatrix} , \\ ; \end{Bmatrix} \text{[print-element]} \right] \ldots$$

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	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
A1xx	compressed alphanumeric format

where:

xx = field width in binary (xx > 0)
y = field width in binary (< y <=4)
d = implied decimal position in binary

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.

Numeric Fields

ASCII free format (10xx)

The data can be any valid BASIC representation of a number; spaces are ignored.

s	d	d	...	d	.	d	d	...	d	E	s	d	d

sign — mantissa — exponent

where:

s = sign (ASCII + or -), optional
d = ASCII digit
1 < number of mantissa digits ≤ 13
decimal point optional
exponent optional (one or two digits)

For the following formats, each number can have up to 13 digits of precision. If there are more than 13 significant digits in a numeric value, the exponent of the number is appropriately adjusted. Leading zeros are ignored.

- ASCII integer format (2dxx)

s	d	d	...	d

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	...	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)	Results in an error because
:110 $UNPACK (F = F$) B$ TO X	B$ contains more than 13 digits.
:100 F$ = HEX(2015)	
:110 $UNPACK (F = F$) B$ TO X	X = 1.234567890123E19

```
Statements                          Results

: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 (A1xx)

The $UNPACK Compressed Alphanumeric Format is used to decompress alphanumeric data compressed with $PACK format A1xx. 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

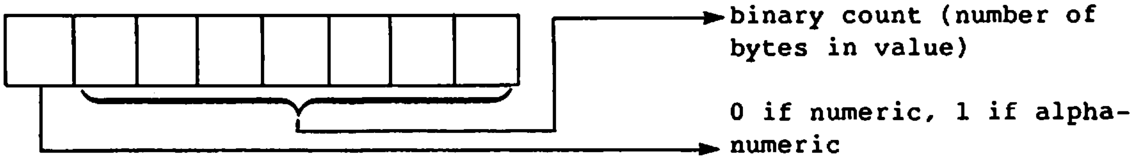
Data stored in the standard Wang 2200 disk record format can be unpacked by the internal form of $UNPACK. Data records that have been saved on a disk platter by either the DATASAVE DC or the DATASAVE DA statement are stored in this format. Data values are sequentially read from the buffer and stored 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_{16}	01_{16}	SOV	value	SOV	value	SOV	value	EOB

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.


binary count (number of bytes in value)
0 if numeric, 1 if alpha-numeric

- 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_L	$e_H d$	dd	dd	dd	dd	dd	dd

where:

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

$e_L e_H$ = exponent (2 digits)

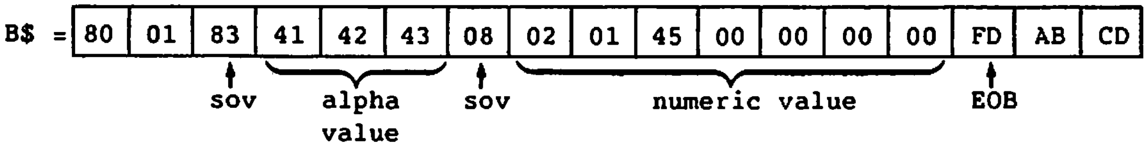
d = mantissa digit (always 13)

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

<u>Note</u>: *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.



B$ =	80	01	83	41	42	43	08	02	01	45	00	00	00	00	FD	AB	CD

```
: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) A$( )
DATASAVE AC #1, A$( )
```

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) "Datal.Dat"
        DATASAVE AC OPEN T#3, "Datal.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:

                                                    BEG
      DBACKSPACE [file#,]
                                                    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

WANG

Customer Comment Form

Publication Number 700-4080E

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?	□	□	□	□	□
Readability — Is the manual easy to read and understand?	□	□	□	□	□
Clarity — Are the instructions easy to follow?	□	□	□	□	□
Examples — Were they helpful, realistic? Were there enough of them?	□	□	□	□	□
Organization — Was it logical? Was it easy to find what you needed to know?	□	□	□	□	□
Illustrations — Were they clear and useful?	□	□	□	□	□
Physical Attractiveness — What did you think of the printing, binding, etc?	□	□	□	□	□

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.

All comments and suggestions become the property of Wang Laboratories, Inc.

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

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

Fold

WANG

To Order by Phone, Call:
1-(800)225-0234
Telex 172108

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 □
No □

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

□ 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	
LocalStateTax	
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

Printed in U.S.A. 700-4080E.02 8-89