# THE WANG PROFESSIONAL COMPUTER

## PC/2200 BASIC-2 Compatibility Guide

WANG

# The Wang Professional Computer

## PC/2200 BASIC-2 Compatibility Guide

**First Edition — May 1984**

**715-0005**

WANG

**WANG LABORATORIES, INC.,** ONE INDUSTRIAL AVENUE, LOWELL, MA 01851 • TEL. (617) 459-5000, TWX 710-343-6769, Telex 94-7421

## 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.

# PREFACE

The PC/2200 BASIC-2 Compatibility Guide provides information on using the BASIC-2 language on the Professional Computer (PC) and the 2200. 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 PC 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, { }, 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.

CONTENTS

CHAPTER 1 GENERAL I/O STATEMENTS

1.1 Introduction ..... 1-1
1.2 $IF ON/OFF ..... 1-2
1.3 $GIO ..... 1-4
Comment Parameter ..... 1-4
Device-Address Parameter ..... 1-5
Microcommand-Sequence Parameter ..... 1-5
Condition Code ..... 1-6
Specification of a Microcommand Sequence ..... 1-7
Registers Parameter ..... 1-8
Data Buffer Parameter ..... 1-8
Character Count ..... 1-10
Terminating Multicharacter I/O Operations ..... 1-10
Output Examples of $GIO ..... 1-12
Input Examples of $GIO ..... 1-14

CHAPTER 2 INFORMATION FOR THE 2200 BASIC-2 USER

2.1 Line Number Range ..... 2-1
2.2 Differences in Key Implementation ..... 2-1
2.3 Variable Name Length ..... 2-1
2.4 Arrays ..... 2-2
2.5 Scrambled (Protected) Programs ..... 2-2
2.6 PC BASIC-2 Statements Not Part of 2200 BASIC-2 ..... 2-2
2.7 2200 BASIC-2 Statements Not Part of PC BASIC-2 ..... 2-2
2.8 The LOAD Statement ..... 2-3
2.9 ON ERROR...GOTO Statement ..... 2-3
2.10 Statement Differences ..... 2-4

INDEX ..... Index-1

TABLES


| | | |
|---|---|---|
| Table 1-1 | Legend | 1-16 |
| Table 1-2 | Summary of Microcommand Categories | 1-18 |
| Table 1-3 | Single Address Strobe | 1-19 |
| Table 1-4 | Control Microcommands | 1-20 |
| Table 1-5 | Single-Character Output Microcommands | 1-23 |
| Table 1-6 | Single-Character Input Microcommands | 1-25 |
| Table 1-7 | Multicharacter Output Microcommands | 1-26 |
| Table 1-8 | Valid CHECK T Codes for Table 1-7 | 1-27 |
| Table 1-9 | Valid LEND Codes for Table 1-7 | 1-27 |
| Table 1-10 | Multicharacter Input Microcommands | 1-28 |
| Table 1-11 | Valid CHECK T Codes for Table 1-10 | 1-29 |
| Table 1-12 | Valid LEND Codes for Table 1-10 | 1-29 |
| Table 1-13 | Register Usage | 1-30 |
| Table 1-14 | Summary of Microcommands for Use with PC BASIC-2 | 1-31 |
| Table 2-1 | Comparison of the General Instruction Sets | 2-4 |
| Table 2-2 | Comparison of the System Commands | 2-11 |
| Table 2-3 | Comparison of I/O Instructions | 2-13 |
| Table 2-4 | Comparison of 2200MVP Instructions | 2-17 |

# 1

# GENERAL I/O STATEMENTS

# CHAPTER 1
# GENERAL I/O STATEMENTS

## 1.1 INTRODUCTION

The general I/O statements, $IF ON/OFF and $GIO, were included as part of 2200 BASIC-2 to access peripherals that could not be accessed by other BASIC-2 statements. The $IF ON/OFF statement in PC BASIC-2 is restricted to the screen and keyboard. The $GIO statement is implemented in PC BASIC-2 primarily for compatibility with existing BASIC-2 programs; $GIO can be used with screen and keyboard I/O only.

NOTE:
Because $GIO and $IF ON/OFF are included in PC BASIC-2 primarily to maintain compatibility with existing 2200 BASIC-2 programs, you should avoid using these statements when writing new PC BASIC-2 programs.

The following pages explain the full syntax of these statements as provided with 2200 BASIC-2. The restricted subset of $GIO operations that can be used with PC BASIC-2 appears in Table 1-14.

## 1.2 $IF ON/OFF

Format:

```
                  ⎧ ON  ⎫ ⎡ device-address, ⎤
$IF               ⎨     ⎬ ⎢                ⎥   line-number
                  ⎩ OFF ⎭ ⎣ file-number,   ⎦
```

The $IF ON/OFF statement determines the ready/busy status of a given device attached to the system and branches when ready or busy, depending upon which form of the statement is used.

The $IF ON statement tests the device ready/busy signal for the ready condition. If device ready is sensed, the program branches to the line specified in the $IF ON statement. If device busy is sensed, program execution continues at the next statement.

Conversely, the $IF OFF statement tests the ready/busy signal for device busy. If device busy is sensed, the program branches to the specified line. If device ready is sensed, program execution continues at the next statement.

PC BASIC-2 restricts the use of $IF ON/OFF to the keyboard (address /001) and the screen (address /005). The keyboard is ready if a key has been pressed but has not yet been input by PC BASIC-2; the keyboard is busy if no key has been pressed. The screen is always ready.

The following methods specify a device-address in a $IF ON/OFF statement:

- Direct address specification using a slash followed by a 3-hexadecimal-digit device-address

```
10 $IF OFF/001, 1000
```

- Indirect address specification using a file-number to which the desired device-address has been previously assigned

```
10 SELECT #3/001
20 $IF ON #3, 250
```

  A file-number identifies the slot in the Device Table to which the address of the I/O device to be tested has been assigned with the SELECT statement. (Refer to The Professional Computer BASIC-2 Language Guide.)

- Omitting an address, thereby implying that the device-address currently selected for TAPE-class operations should be used

```
10 SELECT TAPE/001
20 $IF ON 200
```

Examples of valid syntax:

```
$IF ON/001, 450

SELECT #7/001
$IF ON #7, 100

SELECT TAPE/001
$IF ON 400
```

## 1.3 $GIO

Format:

```
                          [ device-address[,] ]
$GIO [comment]            [                   ]  (microcommand-sequence
                          [ file-number,      ]

  [,registers]) [buffer [;buffer] ...]
```

where:

| | | |
|---|---|---|
| comment | = | A character string, consisting of uppercase letters, digits, and spaces, that identifies the particular operation performed by the $GIO sequence |
| microcommand-sequence | = | An I/O operation defined by using a hex literal, an alpha literal, a string of hex digits, or an alpha-variable |
| registers | = | An alpha-variable whose individual bytes (registers) store special characters and error/status information; dimensioned length must be at least 10 bytes |
| buffer | = | An alpha-variable that serves as the data buffer for multiple-character input and output operations |

The $GIO statement uses a series of microcommands represented by a code that is two bytes (four hexadecimal digits) in length to write I/O routines. This code instructs the system to perform one or more specific operations such as move a specified character into a designated alpha variable or read a string of characters from an external device into a buffer memory.

### 1.3.1 Comment Parameter

Since the I/O operation represented by a $GIO statement is not readily identifiable from its microcommand sequence, a descriptive comment inserted in the $GIO statement is helpful when reviewing or revising a program (e.g., WRITE, READ, and CHECK READY). The comment has no functional purpose; the system ignores it during execution of the $GIO statement. The comment can consist of any combination of uppercase letters, digits, and blanks; other characters are illegal in a comment.

### 1.3.2 Device-Address Parameter

The following methods specify a device-address for a $GIO operation:

- Direct address specification using a slash followed by a 3-hexadecimal-digit device-address

```
200 $GIO READ /001 (M$,R$) B$( )
```

- Indirect address specification using a file-number to which the desired device-address has been previously assigned

```
300 SELECT #2/001
310 $GIO READ #2, (M$,R$) B$( )
```

- Omitting an address, thereby implying that the device-address currently selected for TAPE-class operations should be used

```
400 SELECT TAPE/001
410 $GIO READ (M$,R$) B$( )
```

### 1.3.3 Microcommand-Sequence Parameter

Each microcommand in a sequence consists of a 4-hex-digit code of the form hhhh, where h is a hexadecimal digit (0 to 9 or A to F). The first two hex digits in a microcommand code usually identify the type of operation to be performed, for example, single-character output with echo or multicharacter verify. The last two hex digits supply information, for example, a character to be stored or a register containing a character to be transmitted. There is no practical limit on the number of microcommands that can be specified in a single $GIO statement.

The 21 different categories of microcommands available for use in a $GIO statement are of two basic types: I/O microcommands and control microcommands. Refer to Table 1-2 for a summary of the 21 microcommand categories and their functions.

I/O microcommands send one or more characters to an external device or receive one or more characters from an external device. Each I/O microcommand represents a unique signal sequence defining a fundamental I/O operation. For example, the microcommand 400D instructs the system to transmit a HEX(0D) character (carriage return) to the specified external device. Therefore, specifying a sequence of I/O microcommands is equivalent to programming the signal sequence required to perform the desired I/O operation.

Control microcommands provide the capability of programming complete I/O routines, including testing and branching within a single $GIO statement. Control microcommands do not perform input or output operations directly, but they can perform the following important supplementary programming functions:

- Move, compare, and test specified characters, and set the condition code, depending upon the result of a test
- Check error/status information in a register and set the condition code
- Enable and disable the timeout condition
- Enable and disable a delay before sending all output strobes
- Branch to a specified microcommand within the $GIO sequence or terminate $GIO execution on condition code true or false
- Set up the next specified data buffer
- Move characters between the data buffer and the register
- Increment or decrement the specified register or register pair

### 1.3.4 Condition Code

Certain operations within a $GIO sequence set a flag in memory, called the condition code. The condition code has two possible values: true (on) and false (off). Initially, the CPU sets the condition code to false. The condition code remains false until a condition occurs that alters the status. In general, the condition code is automatically set to true by a variety of special conditions, including certain error and termination conditions. Also, control microcommands can test for specified conditions and explicitly set the condition code to true, depending upon the result of the test.

Once the condition code is set to true, execution of the $GIO statement is terminated. An exception is when the next sequential instruction in the microcommand sequence is a branch instruction that tests the status of the condition code and initiates a branch within the $GIO routine. These two special microcommands have the following forms, where "hhh" is the 3-hexadecimal-digit "address" of a microcommand within the microcommand sequence:

Dhhh (branch to hhh if condition code is true)
Ehhh (branch to hhh if condition code is false)

The address of each microcommand represents its displacement from the first microcommand in the sequence. Thus, HEX(000) is the address of the first microcommand in a $GIO statement, and HEX(001) is the address of the second microcommand, and so on.

Example:

If the condition code is true, the following instruction generates a branch to the second microcommand in the $GIO statement (if the condition code is false, a branch does not occur, and the next microcommand in the sequence is executed):

D001

Conversely, if the condition code is false, the following microcommand generates a branch to the sixth microcommand in the $GIO statement (if the condition code is true, a branch does not occur, and the next microcommand in the sequence is executed):

E005

Also, both branch instructions automatically reset the condition code to false.

The branch instructions provide, in conjunction with the condition code, a means of responding to special conditions in an I/O operation without terminating $GIO statement execution. Additionally, branch instructions serve as powerful tools for constructing loops and branches within a $GIO statement, thus facilitating the implementation of sophisticated, custom-tailored I/O routines in a single statement.

### 1.3.5 Specification of a Microcommand Sequence

You can specify a microcommand sequence defining a desired I/O operation either directly or indirectly in a $GIO statement. Indirect specification of microcommand codes offers several advantages, including easier modification and debugging of the microcommand sequence and use of the same sequence in several different $GIO statements. The following methods specify a microcommand sequence:

- Direct specification of the microcommand sequence using HEXdigits

```
10 $GIO (A000, R$) A$
```

- Direct specification of the microcommand sequence using a HEXliteral

```
10 $GIO (HEX(A000), R$) A$
```

- Indirect specification of the microcommand sequence by assigning the microcommand codes to an alpha-variable and specifying the alpha-variable in a $GIO statement

```
10 A$ = HEX(A000)
20 $GIO (A$, R$) B$
```

### 1.3.6 Registers Parameter

The alpha-variable specified as the registers parameter of a $GIO statement serves as a multipurpose memory area where special characters and error/status information are stored. Each byte of the alpha-variable is called a register, and the variable itself is commonly referred to as the register-variable. The dimensioned length of the register-variable must be at least 10 bytes because the system stores special information in bytes (or registers) 8, 9, and 10. The maximum number of registers that can be accessed is 15 (bytes 1 to 15). If the register-variable contains more than 15 bytes, you cannot use the additional bytes as registers.

The registers parameter is optional in a $GIO statement. If it is omitted, the system uses 15 bytes in a reserved section of memory and initializes them to all zeros. In this case, the BASIC-2 program can neither read nor modify the values of particular registers.

The BASIC-2 program uses the registers to store data characters or special termination characters, to store the binary value defining the duration of an implemented delay or timeout condition, or as a counter to control the execution of a loop. By using certain control microcommands, you can compare the contents of two registers, increment or decrement a specified register or register pair by a fixed amount, and test for specific error bits in the error register.

Although all 15 registers are available for use, the system uses several registers to store information during certain operations. In particular, Register 8 stores error/status information (each bit representing a specific error condition), and Registers 9 and 10 maintain a count of the characters processed during a multicharacter operation. During certain multicharacter input and output operations, Registers 5 and 6 store the calculated LRC character and the ENDI character, respectively. Note that if the application program uses any of these Registers (5, 6, 8, 9, 10), the system alters the values of those registers during certain operations. Refer to Table 1-13 for a summary of register usage for each of the 15 registers.

Registers 8, 9, and 10 are automatically initialized to binary zero (HEX(00)) when the $GIO statement begins execution. Subsequently, Registers 9 and 10 are reset to zero whenever the data buffer pointer is set to point to a new buffer by using one of the following control commands: 18hh, 1Ah0, or 1A00. The remaining registers are not initialized automatically and can be assigned an initial value by the BASIC-2 program.

### 1.3.7 Data Buffer Parameter

The data buffer parameter of a $GIO statement consists of one or more alpha-variables used as data buffers. Data buffers are required only for multicharacter input and output operations. They are not required in a $GIO statement restricted to single-character input or output.

You can use alphanumeric-scalar and array-variables as data buffers. The size of the data buffer is equal to the total number of bytes in the defined length of the variable. The defined length can be the entire dimensioned size of the variable or some specified portion of its total size. The STR function can define the length of this specified portion.

If an alphanumeric-array (1- or 2-dimensional) is used as a data buffer, characters are stored into the array or read from the array sequentially, row by row. The entire array is treated as one contiguous string of characters, starting with the first character of the first element. Element boundaries are ignored.

Multiple data buffers must be separated by semicolons (;). The particular data buffer to be used in an I/O operation is indicated by a data buffer pointer. The pointer is automatically set to point to the first buffer in the data buffer sequence when a $GIO statement is executed. The pointer can be set to point to any buffer in the sequence under program control by using one of three control microcommands.

For sequential processing of buffers, the 1A00 control command increments the pointer to the next sequential buffer. A subsequent I/O operation uses that buffer.

For nonsequential processing of buffers, two control microcommands move the pointer to a specified buffer in the data buffer sequence by specifying the address of the buffer to be used. The address of each buffer in the sequence is simply its displacement from the first buffer in the sequence. Thus, HEX(00) is the address of the first buffer, HEX(01) is the address of the second buffer, and so on. The address of the buffer to be used can be specified immediately in an 18hh microcommand, where hh is the address. Alternatively, the address can be specified indirectly as the contents of a register with a 1Ah0 command, where the third HEXdigit specifies the register (1 to 15) containing the buffer address. (Refer to Table 1-4.)

The data buffer pointer is moved only by one of the three microcommands (i.e., 1A00, 18hh, or 1Ah0). The pointer is never moved automatically (i.e., implicitly). Thus, sequential multicharacter I/O commands continue to use the same data buffer until a new data buffer is designated by explicitly moving the pointer with either the 1A00, 18hh, or 1Ah0 command. Data continues to be sequentially sent from or received into the buffer by each microcommand until a termination condition is sensed or the data pointer is moved to another buffer.

### 1.3.8 Character Count

During a multicharacter input or output operation, the system automatically maintains a total number count of characters sent from or received into a particular buffer. The count is a 2-byte binary number stored in Registers 9 and 10. The low-order eight bits of the count are stored in Register 10, and the high-order eight bits are stored in Register 9. Each time a character is received into or sent from the currently specified buffer, the count is incremented. The count for a particular buffer continues to be incremented by subsequent microcommands that use the same buffer.

The count is reset to binary zero only when the data buffer pointer is moved to a new buffer. Thus, the count for each buffer initially starts with a value of zero and is increased cumulatively to reflect the total number of characters transferred or received by all microcommands using that buffer.

For example, if a buffer is only partially filled by a multicharacter input command, a subsequent multicharacter input command stores data in the remaining unused portion of the buffer and continues to update the count to reflect the total number of characters received in the buffer. If a multicharacter input command continues to input data after the buffer is filled, the count continues to be updated to reflect the total number of characters received. However, the additional characters are lost, and therefore are not stored.

### 1.3.9 Terminating Multicharacter I/O Operations

A simple multicharacter output operation always outputs the total number of characters in the output buffer or the defined portion of the buffer. The output operation terminates when the last character is sent. However, a multicharacter input operation can be terminated when one of the following three conditions occurs:

- An ENDI character is received
- A special termination character is received
- The character count equals the input buffer length

The first way to terminate a multicharacter operation is by "Termination on ENDI Character." Certain I/O devices, including the system keyboard, can send a character with a special ninth bit, called the ENDI bit, in addition to the eight data bits. For example, pressing a special function (SF) key on the system keyboard generates a character with an ENDI bit. If this termination condition is specified, the system checks each incoming character for an ENDI bit and terminates the input operation when one is received. The ENDI character is stored in Register 6. However, only the eight data bits are stored; the ENDI bit is removed prior to storage. The character count does not include the ENDI byte.

Example:

Termination upon reception of a special function key (i.e., by an ENDI bit that is sent) can be achieved in the following manner:

```
60 $GIO /001 (C320, R$) A$
```

The system checks each incoming character to determine if it carries a special ninth bit (ENDI). If an ENDI character is received, the system saves the character in Byte 6 of R$ and terminates the input operation according to LEND; otherwise, the character is saved in A$ and the sequence continues. The character count is maintained in Bytes 9 and 10 of R$.

The second way to terminate a multicharacter operation is by "Termination on Special Character." Any character can be a termination character. The desired character must be stored in Register 1 prior to beginning the input operation. If this termination condition is specified, the system compares each incoming character with the character stored in Register 1. The system terminates the input operation when the specified termination character is received. The special character can either be stored with the data characters in the data buffer or be discarded.

Example:

The following example illustrates a variable length input that is terminated by the input of a specific character:

```
50 STR(R$,1,1) = HEX(0D)
60 $GIO /001 (C310, R$) A$
```

Line 50 stores a carriage-return character in the first byte of R$. The third digit of the microcommand instructs the system to check Byte 1 of R$ after each character is stored and to terminate the input sequence according to LEND if the character received is the same as the character stored in R$. If the character does not match, the sequence is repeated. The program should check Bytes 9 and 10 of R$ to determine how many bytes were actually received. For example, the following statement calculates the number of characters input in the $GIO sequence:

```
70 N = VAL(STR(R$,9,2),2)
```

The final way to terminate a multicharacter operation is by "Termination on Character Count." Most multicharacter input commands can be terminated when the input buffer is filled. If this termination condition is specified, the system compares the character count with the total number of bytes reserved for the buffer after each character is received. The system terminates the input operation when the character count equals the buffer length (i.e., when the buffer is filled).

Example:

The following sequence inputs ten characters:

```
10 $GIO /001 (C340, R$) STR (A$,1,10)
```

This microcommand receives one character at a time from the keyboard and stores it in one of the first ten bytes of A$. After receiving and storing each character, the third digit of the microcommand instructs the system to determine if the buffer is full. If so, the input sequence terminates according to the last digit in the microcommand (LEND).

If the character count is not specified as a termination condition, the number of characters received prior to termination can exceed the available buffer space. In this case, an error bit is set in the error register (Register 8), and the excess characters are received but not stored. Whether or not these characters are stored, the count continues to be updated to reflect the total number of characters received.

You can specify one, two, or all three termination conditions in a single multicharacter input command. If multiple conditions are specified, the input operation terminates when any one of the specified termination conditions are satisfied. The order in which termination conditions are checked by the system are defined as follows:

1. ENDI Character
2. Special Termination Character
3. Character Count

If multiple termination conditions are specified, the system sets a bit in Register 8 to indicate which condition caused termination. Subsequently, the program can check Register 8 to determine which of the specified termination conditions actually caused the input operation to terminate. If only one termination condition is specified in the input command, the system does not set a bit in Register 8.

You can select combinations of termination conditions and LEND sequences by varying the last two digits of the C3t1-type microcommands according to the desired combination. For instance, the C330 microcommand terminates the input sequence when either a special character or an ENDI bit is received.

### 1.3.10 Output Examples of $GIO

The following examples illustrate the use of different $GIO routines to output one or more characters to a device. Usually, standard BASIC-2 statements such as PRINT or LIST display information to the screen, rather than the $GIO statement. To simplify the illustration of these routines, the following examples use the screen as the output device.

Example:

The following $GIO sequence uses Immediate mode output microcommands to display the word WANG on the CRT screen:

```
$GIO /005 (4057 4041 404E 4047)
```

Most simple output devices indicate their readiness to receive a byte by setting their ready/busy signal on the I/O bus to ready. The first byte of the first microcommand (40) waits for this ready signal (WR) and then outputs the character indicated by the HEXcode in the second byte of the microcommand (i.e., 57, which is the character W). Data output is accomplished with a single OBS for each character.

Since specifying each character in an individual microcommand is frequently an inefficient method of outputting data, the data is usually output from a variable. This method allows you to use one $GIO sequence for all output of this type simply by changing the value of the variable during program execution. Indirect $GIO microcommands are used to change the value of the variable that holds the data.

Example:

The following program displays the word WANG on the screen by using variables with the microcommand:

```
10 R$ = "WANG"
20 $GIO /005 (4210 4220 4230 4240, R$)
```

The first byte of each microcommand (42) waits for device ready. Then an OBS sends the byte of R$ indicated by the second byte of the microcommand (e.g., 10 sends the first byte, which is the character W).

Long data strings are output using multicharacter output commands; the data to be output is stored in the data buffer.

Example:

The following $GIO statement displays the first 100 bytes of A$( ):

```
100 $GIO /005 (A000) STR(A$( ),1,100)
```

The STR function indicates that not all of A$( ) is to be displayed, just the first 100 bytes.

### 1.3.11 Input Examples of $GIO

The following examples illustrate $GIO usage for data input from a single device. Usually the keyboard is accessed with the BASIC-2 statements KEYIN, INPUT, and LINPUT, rather than the $GIO statement. The following examples use the keyboard as the input device to provide examples that illustrate typical programming situations:

Example:

The following $GIO sequence uses single character input microcommands to receive three characters from the keyboard. The first three bytes of R$ contain the three characters input from the keyboard.

```
10 $GIO /001 (8701 8702 8703, R$)
```

If more than a few characters are to be input, multicharacter input commands should be used.

Example:

The following example accepts a string of characters from the keyboard and displays each character on the screen as it is entered.

```
10 $GIO (010D 7101 8702 7105 4220 1B22 1C12 E001, R$) A$
```

The microcommand sequence in Line 10 works in the following manner:

- The first microcommand (010D) stores the character whose HEXcode is the last byte of the microcommand in the register specified by the second digit of the microcommand. In this case, a carriage return (HEX(0D)) is stored in byte 1 of R$.
- The second microcommand (7101) enables the device whose address is equal to the second byte of the microcommand (device type is omitted). In this example, the keyboard is now enabled (address 01).
- The third microcommand (8702) waits for the device to be ready and then receives an IBS (and, hence, one character) from the keyboard, storing it in the register specified by the last digit of the command.
- The fourth microcommand (7105) enables the device whose address is equal to the second byte of the microcommand. In this example, the screen is now enabled (address 05).
- The fifth microcommand (4220) waits for screen ready and performs an OBS, sending the character in Register 2 back to the screen.

- The sixth microcommand (1B22) takes the character in Register 2, places it in the data buffer (A$), and increments the count maintained in Registers 9 and 10. Additionally, this command sets the condition code and terminates the $GIO if the data buffer is already full.

- The seventh microcommand (1C12) compares the character in Register 1 to that in Register 2 and sets the condition code if the two are equal.

- The eighth microcommand (E001) tests the condition code. If it is false, a branch to the second microcommand takes place (001); otherwise, the sequence is ended.

Thus, the sequence can be terminated if either the data buffer becomes full or a carriage return is received. The data buffer A$ contains the characters received (including the carriage return), and the count is stored in Registers 9 and 10.

Tables 1-1 through 1-13 show the entire $GIO microcommands set for 2200 BASIC-2. The shaded areas indicate recommended microcommands. Table 1-14 shows the restricted set of microcommands that can be used with PC BASIC-2.

Table 1-1. Legend[a]

| Mnemonic | Operation |
|---|---|
| ABS | CPU sends an Address Bus Strobe with an immediate or indirect address to disable the current address and enable the specified address. |
| CBS | CPU sends a Control Bus Strobe to the enabled device. |
| CHECK ENDI | CPU checks for ENDI condition. If the ENDI bit is sent, the CPU saves the byte in Register 6 (rather than Register r) and sets the 20 bit of Register 8. |
| CHECK $T_1$ | CPU checks for ENDI and Special Character termination conditions, and then proceeds according to Code t. |
| CHECK $T_2$ | CPU checks for both Special Character and full buffer termination conditions, then proceeds according to code t. |
| CPB | CPU sets its input ready/busy signal to ready. |
| DATAOUT | CPU sends next character from $GIO data buffer, then increments the count in Bytes 9 and 10. |
| ECHO | The received character is echoed (with either OBS or CBS). |
| IBS | CPU awaits input strobe from enabled device. If the wait is greater than 8 milliseconds, an error results. |
| IMM | Immediate character is HEX($h_1h_2$), specified by the microcommand. |
| IND | Indirect character is in the register specified by r. |
| LEND | CPU executes the LRC end sequence specified by l. |
| OBS | CPU sends an Output Bus Strobe to the enabled device. |
| SAVE | CPU saves received character in the register specified by r. |
| SAVE DATA | CPU saves received character in the next location of the data buffer, then increments the count. |
| SAVE LRC | CPU saves calculated LRC character in Register 5. |
| SEND LRC | CPU sends calculated LRC character to enabled device. |

[a]Mnemonics used to describe signal sequences for I/O microcommands

Table 1-1. Legend[b] (continued)

| Mnemonic | Operation |
|---|---|
| SET CC | CPU terminates microcommand and sets condition code if specified condition exists. |
| VERIFY | CPU compares received character. If unequal, the CPU sets the echo-verify error bit (Bit 04 in Register 8) to 1. |
| WR | CPU awaits ready signal from enabled device. If the wait is greater than 1 millisecond, a break point can result. |
| W5 | CPU waits (5 microseconds) until OBS or CBS is complete. |
| **Symbol** | **Digit in That Position Represents** |
| a | Portion of 12-bit microcommand address |
| c | Microcode command specification |
| d | Delay specification |
| h | HEX digit |
| l | LEND code |
| r | Register |
| t | CHECK-T code |

[b]Mnemonics used to describe signal sequences for I/O microcommands

Table 1-2. Summary of Microcommand Categories

| Code | | Operation | Refer To |
|---|---|---|---|
| $7ch_1h_2$ | (c = 1, 3, 6) | Single address strobe | Table 1-3 |
| $0rh_1h_2$ | | Control -- store immediate | Table 1-4 |
| $1h_1h_2h_3$ | | Control -- general | |
| $75h_1h_2$ | | Control -- delay immediate | |
| 77r0 | | Control -- delay immediate | |
| $Da_1a_2a_3$ | | Branch control | |
| $Ea_1a_2a_3$ | | Branch control | |
| $4ch_1h_2$ | | Single character output | Table 1-5 |
| $5ch_1h_2$ | | Single character output with acknowledge | |
| $6ch_1h_2$ | | Single character output with echo | |
| $8ch_1h_2$ | (c = 6, 7) | Single character input | Table 1-6 |
| $8ch_1h_2$ | (c = 0-3, 8-B) | Single character input with verify | |
| $9ch_1h_2$ | | Single character input with echo | |
| Ac01 | | Multicharacter output | Table 1-7 |
| Bctl | (c = 0, 1, 4, 5) | Multicharacter output with acknowledge | |
| Bctl | (c = 2, 3, 6, 7) | Multicharacter output with echo | |
| Bctl | (c = 8, 9, C, D) | Multicharacter output with each character requested | |
| BAtl | | Multicharacter verify | |
| Cctl | (c = 2, 3, 6, 7) | Multicharacter input | Table 1-10 |
| Cctl | (c = 0, 1, 4, 5) | Multicharacter input with echo | |
| Cctl | (c = 8-F) | Multicharacter input with each character requested | |

NOTE:
A programmer can use all codes listed in Tables 1-3 through 1-12. However, shaded areas indicate recommended codes for first-time users.

Table 1-3. Single Address Strobe

| Code | Signal Sequence | Verify Character | Character to be Saved |
|---|---|---|---|
| $71h_1h_2$ | ABS/IMM | HEX($h_1h_2$) | |
| 73r0 | ABS/IND | From Register r | |
| 760r | STATUS REQUEST | | In Register r |

NOTE:
Codes in this category can be used repeatedly in a sequence to disable the current device address and enable another.

Table 1-4. Control Microcommands

| Code | Operation |
|---|---|
| $0rh_1h_2$ | Store immediate second character (byte), HEX($h_1h_2$) in Register r. |
| 1000 | Set condition code true. |
| 1010 | Set condition code if device is ready. |
| 1020 | Wait for device ready (with timeout). |
| 1200 | Disable previously set delay/timeout condition. |
| $11r_1r_2$ | Move contents of Register $r_1$ to Register $r_2$. |
| 12r1 | Set a coarse delay before each subsequent OBS or CBS. The length of the delay in units of 50 microseconds is specified by a 2-byte binary value stored in Registers r and r+1 (where $1 \leq r \leq 14$). Maximum delay = HEX(FFFF), which is approximately 3.3 seconds. |
| 12r2 | Set a timeout prior to checking each subsequent device ready signal (for output operations) or input strobe (for input operations). The interval in units of 1 millisecond is specified by a 2-byte binary value stored in Registers r and r+1 (where $1 \leq r \leq 14$). Maximum timeout interval = HEX(FFFF), which is approximately 65.6 seconds. If a timeout interval is exceeded, set condition code and set error bit (Bit 10) in Register 8. (The 2200 multiuser operating system restricts timeout to 15 milliseconds for IBS timeout.) |
| $13d_1d_2$ | Set a fine delay before each subsequent OBS or CBS. The duration of the delay is specified, in units of 5 microseconds, by the binary value of the second byte of the command ($d_1d_2$). HEX(03) = 15 microseconds, HEX(04) = 20 microseconds, and so on. If specified delay is less than actual loop time (currently about 10 microseconds), then reducing value may not matter. Maximum delay = HEX(FF), which is approximately 1.275 milliseconds. (When execution of a $GIO sequence begins, this delay is set to HEX(0A), which is approximately 50 microseconds.) |
| $14r_1r_2$ | If contents of Register $r_1$ do not equal contents of Register $r_2$, set compare error bit (Bit 08, Register 8) to 1. |
| $15r_1r_2$ | If contents of Register $r_1$ do not equal contents of Register $r_2$, set compare error bit (Bit 08, Register 8). Then, if compare error bit is set, set condition code. |

Table 1-4. Control Microcommands (continued)

| Code | Operation |
|---|---|
| $16a_1a_2$ | If complemented status code (Register 8) AND $a_1a_2$ does not equal HEX(00), set condition code (i.e., set cc if any bit specified by the mask $a_1a_2$ is equal to 0). |
| $17a_1a_2$ | If status code (Register 8) AND $a_1a_2$ does not equal HEX(00), set condition code (i.e., set cc if any bit specified by the mask $a_1a_2$ is equal to 1). |
| $18a_1a_2$ | Set data buffer pointer to specified buffer in the data buffer sequence. Buffer pointed to is specified by binary value of second byte of command ($a_1a_2$), which represents displacement from first buffer in sequence. Thus, HEX(1800) points to first buffer, HEX(1801) points to second buffer, HEX(1802) points to third buffer, and so on. If the specified buffer does not exist (i.e., not enough buffers), the system signals an error. This command also resets the count (Registers 9 and 10) to 0. |
| 19rc | Increment/decrement binary value stored in Register r or in pair of Registers r and r+1, depending upon value of c.<br><br>c Operation<br><br>0 Increment Register r by 1<br>1 Increment Register r by 2<br>2 Decrement Register r by 2<br>3 Decrement Register r by 1<br>4 Increment Register pair by 1<br>5 Increment Register pair by 2<br>6 Decrement Register pair by 2<br>7 Decrement Register pair by 1<br><br>If a Register pair is incremented/decremented, it is treated as a 2-byte binary value, with the low-order byte in Register r+1. |
| 1Ar0 | Same as $18a_1a_2$, except displacement is obtained from Register r ($r \geq 1$). |
| 1A00 | Increment data buffer pointer to next buffer. Reset data count to 0. |
| $1Br_1$ | Write one byte from data buffer into Register r. Increment data count (Registers 9 and 10). Set condition code if buffer empty, without changing count or storing byte. |

Table 1-4. Control Microcommands (continued)

| Code | Operation |
|---|---|
| $1Br_2$ | Read one byte from Register r into data buffer. Increment data count (Registers 9 and 10). Set condition code if buffer already full, without changing count or storing byte. |
| $1Cr_1r_2$ | Set condition code if Register $r_1$ equals Register $r_2$. |
| $1Dr_1r_2$ | Set condition code if Register pair ($r_1$, $r_1$+1) equals Register pair ($r_2$, $r_2$+1). |
| $1Er_1r_2$ | Set condition code if Register $r_1$ is greater than Register $r_2$ |
| $1Fr_1r_2$ | Set condition code if Register pair ($r_1$, $r_1$+1) is greater than Register pair ($r_2$, $r_2$+1). |
| $75d_1d_2$ | Delay immediately. This command waits from 0 to 255 milliseconds, using an 8-bit count specified by $d_1d_2$, before continuing to the next command. This delay is invoked only once and is unrelated to other delays. |
| 77r0 | Delay immediately. Same as preceding delay, except that count is obtained from Register r. |
| $Da_1a_2a_3$ | Branch to microcommand whose address is specified by $a_1a_2a_3$ if condition code is true. The 12-bit address specified by $a_1a_2a_3$ is displacement from first microcommand in the microcommand sequence. Thus, HEX(D000) branches to first command in sequence if condition code is true, HEX(D001) branches to second command, and so on. If condition code is false, proceed to next microcommand. Reset condition code to false. |
| $Ea_1a_2a_3$ | Branch to microcommand whose address is specified by $a_1a_2a_3$ if condition code is false. If condition code is true, proceed to next microcommand. Reset condition code to false. |

NOTE:
Because 12-bit addresses are used in the branch instructions, there is a limit of 4,096 microcommands (8192 bytes) that can serve as the destination of a branch in a $GIO sequence.

Table 1-5. Single-Character Output Microcommands

| Code | Signal Sequence | Character To Be Sent | Character To Be Saved |
|---|---|---|---|
| | Single Character Output | | |
| $40h_1h_2$ | WR, OBS/IMM | HEX($h_1h_2$) | |
| $41h_1h_2$ | OBS/IMM | HEX($h_1h_2$) | |
| 42r0 | WR, OBS/IND | From Register r | |
| 43r0 | OBS/IND | From Register r | |
| $44h_1h_2$ | WR, CBS/IMM | HEX($h_1h_2$) | |
| $45h_1h_2$ | CBS/IMM | HEX($h_1h_2$) | |
| 46r0 | WR, CBS/IND | From Register r | |
| 47r0 | CBS/IND | From Register r | |
| | Single Character Output with Acknowledge Input | | |
| $50h_1h_2$ | WR, OBS/IMM, W5, CPB, IBS | HEX($h_1h_2$) | |
| $51h_1h_2$ | OBS/IMM, W5, CPB, IBS | HEX($h_1h_2$) | |
| $52r_1r_2$ | WR, OBS/IND, W5, CPB, IBS, SAVE | From Register $r_1$ | Into Register $r_2$ |
| $53r_1r_2$ | OBS/IND, W5, CPB, IBS, SAVE | From Register $r_1$ | Into Register $r_2$ |
| $54h_1h_2$ | WR, CBS/IMM, W5, CPB, IBS | HEX($h_1h_2$) | |
| $55h_1h_2$ | CBS/IMM, W5, CPB, IBS | HEX($h_1h_2$) | |
| $56r_1r_2$ | WR, CBS/IND, W5, CPB, IBS, SAVE | From Register $r_1$ | Into Register $r_2$ |
| $57r_1r_2$ | CBS/IND, W5, CPB, IBS, SAVE | From Register $r_1$ | Into Register $r_2$ |
| | Single Character Output with Echo Input | | |
| $60h_1h_2$ | WR, OBS/IMM, W5, CPB, IBS, VERIFY | HEX($h_1h_2$) | |
| $61h_1h_2$ | OBS/IMM, W5, CPB, IBS, VERIFY | HEX($h_1h_2$) | |
| $62r_1r_2$ | WR, OBS/IND, W5, CPB, IBS, SAVE, VERIFY | From Register $r_1$ | Into Register $r_2$ |
| $63r_1r_2$ | OBS/IND, W5, CPB, IBS, SAVE, VERIFY | From Register $r_1$ | Into Register $r_2$ |

Table 1-5. Single-Character Output Microcommands (continued)

| Code | Signal Sequence | Character To Be Sent | Character To Be Saved |
|---|---|---|---|
| | Single-Character Input with Echo Output | | |
| 64$h_1h_2$ | WR, CBS/IMM, W5, CPB, IBS, VERIFY | HEX($h_1h_2$) | |
| 65$h_1h_2$ | CBS/IMM, W5, CPB, IBS, VERIFY | HEX($h_1h_2$) | |
| 66$r_1r_2$ | WR, CBS/IND, W5, CPB, IBS, SAVE, VERIFY | From Register $r_1$ | Into Register $r_2$ |
| 67$r_1r_2$ | CBS/IND, W5, CPB, IBS, SAVE, VERIFY | From Register $r_1$ | Into Register $r_2$ |
| 68$h_1h_2$ | WR, OBS/IMM, W5, CPB, IBS, VERIFY, SET CC (if VFY bit set) | HEX($h_1h_2$) | |
| 69$h_1h_2$ | OBS/IMM, W5, CPB, IBS, VERIFY, SET CC (if VFY bit set) | HEX($h_1h_2$) | |
| 6A$r_1r_2$ | WR, OBS/IND, W5, CPB, IBS, SAVE, VERIFY, SET CC (if VFY bit set) | From Register $r_1$ | Into Register $r_2$ |
| 6B$r_1r_2$ | OBS/IND, W5, CPB, IBS, SAVE, VERIFY, SET CC (if VFY bit set) | From Register $r_1$ | Into register $r_2$ |
| 6C$h_1h_2$ | WR, CBS/IMM, W5, CPB, IBS, VERIFY, SET CC (if VFY bit set) | HEX($h_1h_2$) | |
| 6D$h_1h_2$ | CBS/IMM, W5, CPB, IBS, VERIFY, SET CC (if VFY bit set) | HEX($h_1h_2$) | |
| 6E$r_1r_2$ | WR, CBS/IND, W5, CPB, IBS, SAVE, VERIFY, SET CC (if VFY bit set) | From Register $r_1$ | Into register $r_2$ |
| 6F$r_1r_2$ | CBS/IND, W5, CPB, IBS, SAVE, VERIFY, SET CC (if VFY bit set) | From Register $r_1$ | Into Register $r_2$ |

Table 1-6. Single-Character Input Microcommands

| Code | Signal Sequence | Verify Character | Character To Be Saved |
|---|---|---|---|
| | Single-Character Input | | |
| 8600 | CPB, IBS | | |
| 860r | CPB, IBS, SAVE | | Into Register r |
| 862r | CPB, IBS, CHECK ENDI + SAVE | | Into Register r |
| 8700 | WR, CPB, IBS | | |
| 870r | WR, CPB, IBS, SAVE | | Into Register r |
| 872r | WR, CPB, IBS, CHECK ENDI + SAVE | | Into Register r or Register 6, if ENDI |
| | Single-Character Input with Verify | | |
| $80h_1h_2$ | CPB, IBS, VERIFY/IMM | HEX($h_1h_2$) | |
| $81h_1h_2$ | WR, CPB, IBS, VERIFY/IMM | HEX($h_1h_2$) | |
| $82r_1r_2$ | CPB, IBS, SAVE, VERIFY/IND | In Register $r_1$ | Into Register $r_2$ |
| $83r_1r_2$ | WR, CPB, IBS, SAVE, VERIFY/IND | In Register $r_1$ | Into Register $r_2$ |
| $88h_1h_2$ | CPB, IBS, VERIFY/IMM, SET CC (if VFY bit set) | HEX($h_1h_2$) | |
| $89h_1h_2$ | WR, CPB, IBS, VERIFY/IMM, SET CC (if VFY bit set) | HEX($h_1h_2$) | |
| $8Ar_1r_2$ | CPB, IBS, SAVE, VERIFY/IND, SET CC (if VFY bit set) | In Register $r_1$ | Into Register $r_2$ |
| $8Br_1r_2$ | WR, CPB, IBS, SAVE, VERIFY/IND, SET CC (if VFY bit set) | In Register $r_1$ | Into Register $r_2$ |
| | Single-Character Input with Echo Output | | |
| 920r | CPB, IBS, SAVE, WR, ECHO/OBS | | Into Register r |
| 930r | CPB, IBS, SAVE, ECHO/OBS | | Into Register r |
| 960r | CPB, IBS, SAVE, WR, ECHO/CBS | | Into Register r |
| 970r | CPB, IBS, SAVE, ECHO/CBS | | Into Register r |

Table 1-7. Multicharacter Output Microcommands

| Code | Signal Sequence[c] |
|---|---|
| | Multicharacter Output |
| A001 | (WR, DATAOUT/OBS), LEND |
| A101 | ( DATAOUT/OBS), LEND |
| A201 | 25 microsecond version of A001; no timeout or delay |
| A401 | (WR, DATAOUT/CBS), LEND |
| A501 | ( DATAOUT/CBS), LEND |
| A601 | SCAN DATA BUFFER, CALCULATE LRC, LEND |
| | Multicharacter Output with Acknowledge Input |
| B0t1 | (WR, DATAOUT/OBS, W5, CPB, IBS, CHECK T), LEND |
| B1t1 | ( DATAOUT/OBS, W5, CPB, IBS, CHECK T), LEND |
| B4t1 | (WR, DATAOUT/CBS, W5, CPB, IBS, CHECK T), LEND |
| B5t1 | ( DATAOUT/CBS, W5, CPB, IBS, CHECK T), LEND |
| | Multicharacter Output with Echo Input |
| B2t1 | (WR, DATAOUT/OBS, W5, CPB, IBS, VERIFY, CHECK T), LEND |
| B3t1 | ( DATAOUT/OBS, W5, CPB, IBS, VERIFY, CHECK T), LEND |
| B6t1 | (WR, DATAOUT/CBS, W5, CPB, IBS, VERIFY, CHECK T), LEND |
| B7t1 | ( DATAOUT/CBS, W5, CPB, IBS, VERIFY, CHECK T), LEND |
| | Multicharacter Output with Each Character Requested |
| B8t1 | (CPB, IBS, CHECK T, WR, DATAOUT/OBS), LEND |
| B9t1 | (CPB, IBS, CHECK T, DATAOUT/OBS), LEND |
| BCt1 | (CPB, IBS, CHECK T, WR, DATAOUT/CBS), LEND |
| BDt1 | (CPB, IBS, CHECK T, DATAOUT/CBS), LEND |
| | Multicharacter Verify (of Multicharacter Input) |
| BAt0 | (CPB, IBS, VERIFY, CHECK T) |

[c] A sequence in parentheses is repeated for each character in the data buffer.

Table 1-8. Valid CHECK T Codes for Table 1-7

| Termination Condition[d] | Microcommand | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | B0t1 B1t1 | B4t1 B5t1 | B2t1 B3t1 | B6t1 B7t1 | B8t1 B9t1 | BCt1 BDt1 | BAt0 |
| None (Loop until buffer is done) | 0 | | 0 | | 0 | | 8 |
| Terminate output sequence if verify unequal; set cc | | | 1 | | | | 9 |
| Terminate output sequence if ENDI logic level '1'; set cc | 2 | | 2 | | 2 | | A |
| Terminate output sequence on either condition; set cc | | | 3 | | | | B |

Table 1-9. Valid LEND Codes for Table 1-7

| LRC[e] End Sequence | Microcommand | | | |
|---|---|---|---|---|
| | Act1 | B0t1 through B7t1 | B8t1 BCt1 | B9t1 BDt1 |
| None; go to next microcommand | 0 | 0 | 0 | |
| WR, SEND LRC/OBS, SAVE LRC | 2 | 2 | | |
| SEND LRC/OBS, SAVE LRC | 3 | 3 | | |
| SAVE LRC | 4 | 4 | 4 | |
| WR, SEND LRC/CBS, SAVE LRC | 6 | 6 | | |
| SEND LRC/CBS, SAVE LRC | 7 | 7 | | |

[d]These termination conditions end the microcommand, not the $GIO, and do not set the condition code.

[e]The LRC is the XOR of all bytes transferred to or from the buffer in the present command.

Table 1-10. Multicharacter Input Microcommands[f]

| Code | Signal Sequence |
|---|---|
| | Multicharacter Input |
| C221 | ( CPB, IBS, no timeout or delay, CHECK ENDI, SAVE DATA), LEND |
| C3t1 | (WR, CPB, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| C6t1 | ( CPB, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| C7t1 | (Delay 50 microseconds, CPB, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| | Multicharacter Input with Echo |
| C0t1 | (CPB, IBS, WR, ECHO/OBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| C1t1 | (CPB, IBS, ECHO/OBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| C4t1 | ( IBS, WR, ECHO/OBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| C5t1 | (CPB, IBS, ECHO/OBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| | Multicharacter Input with Each Character Requested |
| C8t1 | (WR, OBS, W5, CPB, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| C9t1 | ( OBS, W5, CPB, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| CAt1[g] | (CPB, WR, OBS, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| CBt1[g] | (CPB, OBS, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| CCt1 | (WR, CBS, W5, CPB, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| CDt1 | ( CBS, W5, CPB, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| CEt1[g] | (CPB, WR, CBS, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |
| CFt1[g] | (CPB, CBS, IBS, CHECK $T_1$, SAVE DATA, CHECK $T_2$), LEND |

[f] A sequence in parentheses is repeated until a valid termination condition occurs. In each command, the t-value is the termination code; the l-value the LEND code.

[g] The four indicated microcommands cannot be used with a 2200 multiuser operating system. An illegal microcommand error results.

Table 1-11. Valid CHECK T Codes for Table 1-10

| t | Termination Conditions[h] (order of checking from left to right) | | |
|---|---|---|---|
| | ENDI-level = 1 when Character Received | Special Character Received Matches Character in Register 1 | Character Count Equals Buffer Length |
| 0 | No action | Check; save in buffer, include in LRC and count | No action |
| 1 | No action | Check; do not save | No action |
| 2 | Check; save in Register 6 | No action | No action |
| 3 | Check; save in Register 6 | Check; do not save | No action |
| 4 | No action | No action | Check |
| 5 | No action | Check; do not save | Check |
| 6 | Check; save in Register 6 | No action | Check |
| 7 | Check; save in Register 6 | Check; do not save | Check |

Table 1-12. Valid LEND Codes for Table 1-10

| 1 | LRC[i] End Sequence |
|---|---|
| 0 | None: go to next microcommand |
| 1 | Calculate LRC and save |
| 2 | Calculate LRC, save, compare with ENDI character, and set LRC error bit. (Use only if t = 2.) |

[h]These termination conditions end the microcommand, not the $GIO, and do not set the condition code.

[i]The LRC is the XOR of all bytes transferred to or from the buffer in the present command.

Table 1-13. Register Usage

| Register (Byte) | Bit Position[J] | Use |
|---|---|---|
| 0 | All | Dummy location; if written to, data is lost; if read, data is always 00 |
| 1 | All | General-purpose or special termination character |
| 2, 3, 4 | All | General-purpose |
| 5 | All | General-purpose or automatic storage of an LRC character |
| 6 | All | General-purpose or automatic storage of an ENDI-level = 1 character |
| 7 | All | General-purpose |
| 8 | 01<br>02<br>04<br>08<br>10<br>20<br>40<br>80 | 1 = Buffer overflow<br>1 = LRC error<br>1 = Echo/Verify error<br>1 = Compare error<br>1 = Timeout<br>1 = ENDI-level termination<br>1 = Special character termination<br>1 = Count termination |
| 9, 10 | All | Automatic storage of the count of transferred characters to or from the currently selected data buffer |
| 11, 12, 13, 14, 15 | All | General-purpose |

NOTE
When $GIO begins execution, the system sets Registers 8, 9, and 10 to 0. Whenever an 18$a_1a_2$, 1A00, or 1Ar0 command is executed, the system resets Registers 9 and 10 to 0.

[J]Bit position labels for status code (Register 8) are as follows:

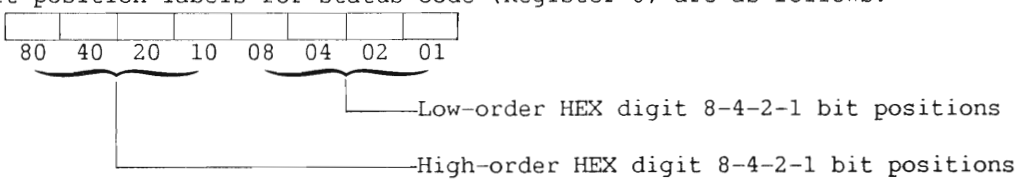

Table 1-14 Summary of Microcommands for Use with PC BASIC-2

| Code | Operation | Refer To |
|---|---|---|
| $7ch_1h_2$ (c = 1, 3, 6) | Single address strobe | Table 1-3 |
| $0rh_1h_2$ | Control -- store immediate | Table 1-4 |
| $1h_1h_2h_3$ | Control -- general | |
| $75h_1h_2$ | Control -- delay immediate | |
| 77r0 | Control -- delay immediate | |
| $Da_1a_2a_3$ | Branch control | |
| $Ea_1a_2a_3$ | Branch control | |
| $40h_1h_2$, $42h_1h_2$ | Single character output | Table 1-5 |
| $8ch_1h_2$ (c = 6, 7) | Single character input | Table 1-6 |
| Ac01 | Multicharacter output | Table 1-7 |
| Cctl (c = 2, 3, 6, 7) | Multicharacter input | Table 1-10 |

PC BASIC-2 $GIO is restricted to the keyboard (/001) and the screen (/005).

# 2

# INFORMATION FOR THE 2200 BASIC-2 USER

# CHAPTER 2
# INFORMATION FOR THE 2200 BASIC-2 USER

This chapter is included primarily for the benefit of those users already familiar with the BASIC-2 language as used on the 2200. It summarizes the language incompatibilities between 2200 BASIC-2 and PC BASIC-2 and describes the implementation of individual statements in Wang BASIC (BASIC as used on the 2200B, 2200C, 2200S, and 2200T), 2200 BASIC-2, and PC BASIC-2.

Programs transferred between the two systems may require adjustments. BASIC-2 users on both the 2200 and the PC should be aware of language differences described in the following sections.

## 2.1 LINE NUMBER RANGE

The range of legal line numbers in 2200 BASIC-2 is from 0 to 9999. In PC BASIC-2, the line number range is from 0 to 999999.

## 2.2 DIFFERENCES IN KEY IMPLEMENTATION

Several keys on 2200 series keyboards, such as CLEAR and LOAD, do not have counterparts on the Wang PC keyboard. However, you can obtain the functions these keys provide in 2200 BASIC-2 through combinations of keys on the PC keyboard. Refer to The Professional Computer BASIC-2 Language Guide for more information.

## 2.3 VARIABLE NAME LENGTH

In 2200 BASIC-2, a variable name consists of a single uppercase letter, optionally followed by a single numeral. In PC BASIC-2, variable names can contain up to 63 characters. The first character is always a capital letter; the remaining characters can include lowercase letters, digits, and periods.

## 2.4 ARRAYS

2200 BASIC-2 arrays contain either one or two dimensions. In PC BASIC-2, an array can contain up to 255 dimensions. The following restrictions apply to PC BASIC-2 array size:

- An array cannot contain more bytes than are in memory.
- A single dimension cannot contain more than 65535 elements.

## 2.5 SCRAMBLED (PROTECTED) PROGRAMS

The PC BASIC-2 interpreter cannot load scrambled programs (2200 programs saved with the ! or P options of the SAVE DC or SAVE DA commands). Also, the ! and P options of these commands have no effect when saving a program on a PC BASIC-2 disk image.

## 2.6 PC BASIC-2 STATEMENTS NOT PART OF 2200 BASIC-2

2200 BASIC-2 does not support the PC BASIC-2 RENAME and SOUND statements. In addition, some statements differ in their implementation under 2200 BASIC-2. Refer to the tables in Section 2.10 for a full explanation.

## 2.7 2200 BASIC-2 STATEMENTS NOT PART OF PC BASIC-2

PC BASIC-2 does not support the following 2200 BASIC-2 statements:

- LIST I
- SELECT @PART
- SELECT TC
- SELECT ON/OFF
- $ALERT
- $CLOSE
- $DISCONNECT
- $FORMAT DISK
- $IF ON/OFF
- $MSG
- $OPEN
- $RELEASE PART
- $RELEASE TERMINAL

If these statements exist in a PC BASIC-2 program, they will either be ignored or will generate an error when executed. In addition, some statements differ in their implementation under PC BASIC-2. Refer to the tables in Section 2.10 for a full explanation.

## 2.8 THE LOAD STATEMENT

The PC BASIC-2 LOAD statement can load programs saved in PC format or programs saved on a 2200 system in 2200 format and transferred to a PC disk image. However, the loading time for programs saved in 2200 format is relatively slow because PC BASIC-2 must convert the program to PC format. After loading a program in 2200 format, you should save the program so that it will be in PC format and thus load more quickly.

Programs that are saved by PC BASIC-2 and then transferred to a 2200 system cannot be loaded by 2200 BASIC-2.

## 2.9 ON ERROR...GOTO STATEMENT

For compatibility, PC BASIC-2 includes the Wang BASIC statement ON ERROR...GOTO. However, the minimum size of the variables that receive the error number and the line number on which the error occurred are changed. PC BASIC-2 requires that three bytes be used to receive the error number and six bytes be used to receive the line number.

## 2.10 STATEMENT DIFFERENCES

The following tables provide a statement-by-statement description of the compatibility and differences among Wang BASIC, 2200 BASIC-2, and PC BASIC-2.

Table 2-1. Comparison of the General Instruction Sets

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| ADD | Yes (Statement only) | Yes (Statement and operator) | Yes (Statement and operator) |
| ALL | No | Yes | Yes |
| AND, OR, XOR | Yes (Statement only) | Yes (Statement and operator) | Yes (Statement and operator) |
| BIN | Yes (Statement only) | Yes (Statement and function) | Yes (Statement and function) |
| BOOL | Yes (Statement only) | Yes (Statement and operator) | Yes (Statement and operator) |
| COM | Yes | Yes (Variables can specify dimensions) | Yes (Up to 255 dimensions are allowed; 32767 is the maximum subscript value) |
| COM CLEAR | Yes | Yes | Yes |
| CONVERT | Yes | Yes (Extended image) | Yes (Extended image) |
| DAC | No | Yes | Yes |
| DATA | Yes | Yes (Hex values are legal) | Yes (Hex values are legal) |
| DATE | No | Yes | Yes |
| DEFFN | Yes | Yes | Yes |
| DEFFN' | Yes | Yes | Yes (Integer that follows prime can now be up to 32767) |

Table 2-1. Comparison of the General Instruction Sets (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| DEFFN @PART | No | Yes | No |
| ERR | No | Yes | Yes |
| ERROR | No | Yes | Yes |
| EXP | Yes | Yes | Yes |
| FIX | No | Yes | Yes |
| FOR | Yes | Yes | Yes |
| $FORMAT | No | Yes | Yes |
| $GIO | Yes | Yes (Includes new microcommands and faster transfer speeds) | Yes (Limited implementation; refer to Chapter 1) |
| GOSUB | Yes | Yes | Yes |
| GOSUB' | Yes | Yes | Yes (Integer that follows prime can be up to 32767) |
| GOTO | Yes | Yes | Yes |
| HEX | Yes | Yes | Yes |
| HEXPACK | No | Yes | Yes |
| HEXPRINT | Yes | Yes (Replaced by PRINT HEXOF) | Yes |
| HEXUNPACK | No | Yes | Yes |
| IF END THEN | Yes | Yes (A statement after THEN is allowed and an ELSE clause is supported) | Yes |
| $IF ON | Yes | Yes | Yes (Use restricted to screen and keyboard) |
| $IF OFF | No | Yes | Yes (Use restricted to screen and keyboard) |

Table 2-1. Comparison of the General Instruction Sets (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| IF/THEN | Yes | Yes (Multiple conditions are allowed, a statement after THEN is allowed, and an ELSE clause is supported) | Yes (Parentheses can be used in conditional expression to alter precedence; NOT operator allowed in relational expressions) |
| Image (%) | Yes | Yes (May be specified as literal in PRINTUSING statement or as value of an alpha-variable; expanded numeric formats) | Yes (As in 2200 BASIC-2) |
| INIT | Yes | Yes | Yes |
| INT | Yes | Yes | Yes |
| INPUT | Yes | Yes (Hex literal legal for INPUT message) | Yes (As in 2200 BASIC-2) |
| KEYIN | Yes | Yes (Line-numbers optional, explicit device-address allowed) | Yes (Can only receive characters from keyboard) |
| LET | Yes | Yes (Wider range of alpha expressions allowed) | Yes (As in 2200 BASIC-2) |
| LEN | Yes | Yes | Yes |
| LINPUT | No | Yes | Yes (2200 BASIC-2 concept of edit mode not applicable; question mark is ignored) |
| LOG | Yes | Yes | Yes |
| LGT | No | Yes | Yes |

Table 2-1. Comparison of the General Instruction Sets (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| MAT Addition | Yes | Yes | Yes (Up to 255 dimensions allowed) |
| MAT CON | Yes | Yes | Yes (Up to 255 dimensions allowed) |
| MAT CONVERT | Yes | Yes (Replaced by MAT MOVE variation) | Yes (Up to 255 dimensions allowed) |
| MAT COPY | Yes | Yes (Alpha-scalar-variables can be specified) | Yes (Up to 255 dimensions allowed) |
| MAT Equality | Yes | Yes | Yes (Up to 255 dimensions allowed) |
| MAT IDN | Yes | Yes | Yes (Array must be square) |
| MAT INPUT | Yes | Yes | Yes |
| MAT INV | Yes | Yes (Normalized determinant returned; error messages are suppressed if determinant specified) | Yes (As in 2200 BASIC-2) |
| MAT MERGE | Yes | Yes | Yes |
| MAT MOVE | Yes | Yes (Automatic alpha-to-numeric, numeric-to-alpha conversions are performed; locator array optional) | Yes (Up to 255 dimensions allowed) |
| MAT Multiplication | Yes | Yes | Yes |
| MAT PRINT | Yes | Yes | Yes |
| MAT READ | Yes | Yes | Yes |
| MAT REDIM | Yes | Yes | Yes |
| MAT Scalar Multiplication | Yes | Yes | Yes |

Table 2-1. Comparison of the General Instruction Sets (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| MAT Subtraction | Yes | Yes | Yes |
| MAT SEARCH | Yes | Yes (Alpha-scalar-variable or a literal can be searched; literal can specify target substring) | Yes (As in 2200 BASIC-2) |
| MAT SORT | Yes | Yes | Yes (Up to 255 dimensions allowed; array having two dimensions can have no array greater than 255) |
| MAT TRN | Yes | Yes | Yes |
| MAT ZER | Yes | Yes | Yes |
| MAX | No | Yes | Yes |
| MIN | No | Yes | Yes |
| MOD | No | Yes | Yes |
| NEXT | Yes | Yes (Multiple indices allowed) | Yes (As in 2200 BASIC-2) |
| NUM | Yes | Yes | Yes |
| ON ERROR GOTO | Yes | Yes (For compatibility with Wang BASIC) | Yes (Alpha-variable to receive error number must be at least three bytes long; alpha-variable to receive line number must be at least six bytes long) |
| ON/GOTO, GOSUB | Yes | Yes (Line-numbers optional; alpha-variable may be used for expression) | Yes (As in 2200 BASIC-2) |
| ON/SELECT | No | Yes | Yes |

Table 2-1. Comparison of the General Instruction Sets (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| PACK | Yes | Yes | Yes |
| $PACK | Yes | Yes (Additional formats) | Yes (As in 2200 BASIC-2) |
| #PI | Yes | Yes | Yes |
| PLOT | Yes | Yes | Yes (Requires medium-resolution graphics board; different syntax) |
| POS | Yes | Yes (Supports last occurrence; target character can be specified by an alpha-variable; can search a literal) | Yes (As in 2200 BASIC-2) |
| PRINT | Yes | Yes (Numeric value with absolute value, which can be expressed in 14 digits is printed in normal format) | Yes |
| PRINT AT | No | Yes | Yes |
| PRINT BOX | No | Yes | Yes (Need a medium-resolution graphics board) |
| PRINT HEXOF | No | Yes | Yes |
| PRINTUSING | Yes | Yes (Expanded image; hex literals legal) | Yes (As in 2200 BASIC-2) |
| PRINTUSING TO | No | Yes | Yes |
| PRINT TAB | Yes | Yes | Yes |
| READ | Yes | Yes | Yes |
| REM | Yes | Yes (Special formatting features added for LIST) | Yes (As in 2200 BASIC-2) |

Table 2-1. Comparison of the General Instruction Sets (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| RENAME | No | No | Yes |
| RESTORE | Yes | Yes (Line-number of DATA statement may be specified) | Yes (As in 2200 BASIC-2) |
| RETURN | Yes | Yes | Yes |
| RETURN CLEAR | Yes | Yes (ALL parameter added) | Yes (As in 2200 BASIC-2) |
| RND | Yes | Yes | Yes |
| ROTATE | Yes | Yes (Supports entire string rotation; left or right rotation) | Yes (As in 2200 BASIC-2) |
| ROUND | No | Yes | Yes |
| SELECT | Yes | Yes | Yes |
| SELECT ERROR | No | Yes | Yes |
| SELECT LINE | No | Yes | Yes |
| SELECT ON, OFF | No | Yes | No (Gives execution phase error) |
| SELECT [NO] ROUND | No | Yes | Yes |
| SGN | Yes | Yes | Yes |
| SOUND | No | No | Yes |
| SPACE | No | Yes | Yes |
| SQR | Yes | Yes (Increased accuracy) | Yes (As in 2200 BASIC-2) |
| STOP | Yes | Yes (Hex literals allowed; line-number can be displayed) | Yes (As in 2200 BASIC-2) |

Table 2-1. Comparison of the General Instruction Sets (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| STR | Yes | Yes (Arrays may be specified; starting byte optional) | Yes (As in 2200 BASIC-2) |
| SUB | No | Yes | Yes |
| TIME | No | Yes | Yes |
| Trig Functions (All) | Yes | Yes | Yes |
| $TRAN | Yes | Yes (Literal can be used as table) | Yes (As in 2200 BASIC-2) |
| UNPACK | Yes | Yes | Yes |
| $UNPACK | Yes | Yes (Additional format) | Yes (As in 2200 BASIC-2) |
| VAL | Yes | Yes (Two-byte conversion supported) | Yes (As in 2200 BASIC-2) |
| VER | No | Yes | Yes |

Table 2-2. Comparison of the System Commands

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| CLEAR | Yes | Yes (Line-numbers optional in CLEAR P) | Yes (Implemented on the 2ND + ERASE keys) |
| CONTINUE | Yes | Yes (Immediate mode statements) | Yes (Implemented on the 2ND + GO TO keys) |
| HALT/STEP | Yes | Yes | Yes (Implemented on the 2ND + STOP keys) |
| LIST S | Yes | Yes (Programmable; title may be specified) | Yes (As in 2200 BASIC-2) |

Table 2-2. Comparison of the System Commands (continued)

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| LIST D | No | Yes | Yes |
| LIST DT | No | Yes | Yes (Different table format; refer to the PC BASIC-2 Language Guide) |
| LIST I | No | Yes | No |
| LIST V | No | Yes | Yes |
| LIST # | No | Yes | Yes |
| LIST' | No | Yes | Yes (Integer following the prime can be up to 32767) |
| RENUMBER | Yes | Yes | Yes (Line numbers up to 999999 are legal; uses comma instead of a dash) |
| RESET | Yes | Yes (CI and CO automatically reselected to keyboard and CRT, respectively) | Yes (Implemented on the 2ND + HELP keys; does not reset a "hung" system) |
| RUN | Yes | Yes (Program execution can start at a specified statement within a line) | Yes (As in 2200 BASIC-2) |
| Special Function Keys | Yes | Yes (INPUT logic used for subroutine entry) | Yes (Special Function Keys 1 to 32 on the PC keyboard correspond to '0 to '31 on the 2200 series keyboards) |
| TRACE | Yes | Yes | Yes |
| TRACE DISK | No | Yes | No |

Table 2-3. Comparison of I/O Instructions

| I/O Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| BACKSPACE (tape drive) | Yes | No (Device not supported) | No (Device not supported) |
| COPY | Yes | No (Replaced by COPY TO) | No |
| COPY TO | No | Yes (Drive-to-drive copy) | Yes (Copy from one disk image to another) |
| DATALOAD (2214 Manual-Feed Card Reader) | Yes | No | No |
| DATALOAD (tape cassette drive) | Yes | No | No |
| DATALOAD (teletype) | Yes | No | No |
| DATALOAD (all other devices) | Yes | Yes | No |
| DATALOAD BA | Yes | Yes (Return-variable optional) | Yes (As in 2200 BASIC-2) |
| DATALOAD BT (2214 Manual-Feed Card Reader) | Yes | No | No |
| DATALOAD BT (tape cassette drive) | Yes | No | No |
| DATALOAD BT (teletype) | Yes | No | No |
| DATALOAD BT (all other devices) | Yes | Yes (Some devices not supported on 2200MVP; refer to the 2200MVP Introductory Manual) | No |
| DATALOAD DA | Yes | Yes (Return-variable optional) | Yes (As in 2200 BASIC-2) |
| DATALOAD DC | Yes | Yes | Yes |
| DATALOAD DC OPEN | Yes | Yes | Yes (No TEMP files) |

Table 2-3. Comparison of I/O Instructions (continued)

| I/O Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| DATASAVE (tape cassette drive) | Yes | No | No |
| DATASAVE (teletype) | Yes | No | No |
| DATASAVE (all other devices) | Yes | Yes (Some devices not supported on 2200MVP; refer to the 2200MVP Introductory Manual) | No |
| DATASAVE BA | Yes | Yes (Return-variable optional) | Yes (As in 2200 BASIC-2) |
| DATASAVE BT (tape cassette drive) | Yes | No | No |
| DATASAVE BT (teletype) | Yes | No | No |
| DATASAVE BT (all other devices) | Yes | Yes (Some devices not supported on 2200MVP; refer to the 2200MVP Introductory Manual) | No |
| DATASAVE DA | Yes | Yes (Return-variable-optional) | Yes (As in 2200 BASIC-2) |
| DATASAVE DC | Yes | Yes | Yes |
| DATASAVE DC CLOSE | Yes | Yes | Yes |
| DATASAVE DC END | Yes | Yes | Yes |
| DATASAVE DC OPEN | Yes | Yes (Syntax for renaming scratched file conforms to SAVE DC) | Yes (Except no TEMP files) |
| DATARESAVE (tape cassette drive) | Yes | No | No |
| DBACKSPACE | Yes | Yes | Yes |
| DSKIP | Yes | Yes | Yes |

Table 2-3. Comparison of I/O Instructions (continued)

| I/O Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| FILE NUMBERS | Yes (0-6) | Yes (0-15) | Yes (0-15) |
| $FORMAT DISK | No | Yes | No |
| LIMITS | Yes | Yes (Status of file may be returned) | Yes (As in 2200 BASIC-2) |
| LIST DC | Yes | Yes (S parameter allowed; available free space returned for each file) | Yes (As in 2200 BASIC-2) |
| LOAD (2214 Manual-Feed Card Reader) | Yes | No | No |
| LOAD (tape cassette drive) | Yes | No | No |
| LOAD (teletype) | Yes | No | No |
| LOAD (all other devices) | Yes | Yes (Some devices not supported on 2200MVP; refer to the 2200MVP Introductory Manual) | Yes |
| LOAD DA | Yes | Yes (Return-variable is optional) | Yes (As in 2200 BASIC-2) |
| LOAD DC | Yes | Yes | Yes |
| LOAD RUN | No | Yes | Yes |
| MOVE | Yes | No (Replaced by MOVE TO) | No |
| MOVE TO | No | Yes (Drive-to-drive move; individual files can be moved) | Yes (Moves files from one disk image to another) |
| MOVE END | Yes | Yes | Yes (Can enlarge a disk image but cannot truncate it) |

Table 2-3. Comparison of I/O Instructions (continued)

| I/O Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| REWIND (tape cassette drive) | Yes | No | No |
| SAVE DA | Yes | Yes (Return-variable is optional) | Yes (As in 2200 BASIC-2 except that ! and P parameters have no effect) |
| SAVE DC | Yes | Yes (Programmable; DC parameter optional; new ! parameter provides more sophisticated program protection capability; ability to automatically delete REMs and spaces from saved program) | Yes (As in 2200 BASIC-2 except that ! and P parameters have no effect) |
| SCRATCH | Yes | Yes | Yes |
| SCRATCH DISK | Yes | Yes | Yes |
| SKIP (tape cassette drive) | Yes | No | No |
| VERIFY | Yes | Yes (Location of bad sector can be returned in a variable without a system error message) | Yes (As in 2200 BASIC-2) |

Table 2-4. Comparison of 2200MVP Instructions

| Instruction | Wang BASIC | 2200 BASIC-2 | PC BASIC-2 |
|---|---|---|---|
| $BREAK statement | No | Yes | Yes (Creates a short delay) |
| $CLOSE statement | No | Yes | No (Statement ignored) |
| DEFFN @PART statement | No | Yes | No (Execution phase error) |
| #ID function | No | Yes | Always returns 0 |
| $INIT statement | No | Yes | Execution of $INIT"SYSTEM" exits the PC BASIC-2 interpreter and returns control to the DOS Command Processor or menu from which PC BASIC-2 was invoked |
| $MSG function | No | Yes | No (Statement ignored) |
| $OPEN function | No | Yes | No (Statement ignored) |
| #PART function | No | Yes | Always returns 1 |
| $RELEASE PART statement | No | Yes | No (Statement ignored) |
| $RELEASE TERMINAL statement | No | Yes | No (Statement ignored) |
| SELECT @PART statement | No | Yes | No (Execution phase error) |
| #TERM function | No | Yes | Always returns 1 |

# INDEX

# INDEX


A

ADD, 2-4
ALL, 2-4, 2-10
Alpha-variable, 1-4
AND, OR, XOR, 2-4
Arrays, 2-2, 2-11

B

BACKSPACE, 2-13
BIN, 2-4
BOOL, 2-4
Branch instructions, 1-7
Buffer, 1-8 to 1-13

C

Character count, 1-10 to 1-12
CHECK READY, 1-4
CLEAR, 2-1, 2-4
COM, 2-4
COM CLEAR, 2-4
Comment, 1-4
Condition Code, 1-6, 1-7
CONVERT, 2-3, 2-4
COPY, 2-7, 2-13
COPY TO, 2-13

D

DAC, 2-4
DATA, 2-4, 2-10
Data Buffer Parameter, 1-8
DATALOAD, 2-13
DATALOAD BA, 2-13
DATALOAD BT, 2-13
DATALOAD DA, 2-13
DATALOAD DC, 2-13
DATALOAD DC OPEN, 2-13
DATASAVE BA, 2-14
DATASAVE BT, 2-14
DATASAVE DA, 2-14
DATASAVE DC CLOSE, 2-14
DATASAVE DC END, 2-14
DATASAVE, 2-14
DATASAVE DC, 2-14
DATASAVE DC OPEN, 2-14
DATE, 2-4
DBACKSPACE, 2-14
DEFFN, 2-4, 2-5
Device-Address Parameter, 1-5
Device Table, 1-2
DSKIP, 2-14

E

ENDI Character, 1-10 to 1-12
ERR, 2-5
ERROR, 2-5, 2-16
EXP, 2-5

F

File number, 1-4, 1-5
FILE NUMBERS, 2-15
FIX, 2-5
FOR, 2-5

G

GOSUB, 2-5
GOSUB', 2-5
GOTO, 2-3

H

HALT, 2-11
HEX, 2-5
HEXPRINT, 2-5
HEXUNPACK, 2-5

I

I/O operation, 1-4
IF END THEN, 2-5
IF/THEN, 2-6
IMAGE, 2-6
INIT, 2-6, 2-17
INPUT, 2-6
INT, 2-6

K

KEYIN, 1-14

L

LEN, 2-6
LET, 2-6
LGT, 2-6
LIMITS, 2-15
Line Number Range, 2-1
LINPUT, 1-14
LIST, 1-12
LIST D, 2-12
LIST DC, 2-15
LIST DT, 2-12
LIST I, 2-12
LIST S, 2-11
LIST V, 2-12
LIST", 1-12, 2-2
LIST #, 1-12, 2-2
LOAD, 2-1 to 2-3
LOAD DA, 2-15
LOAD DC, 2-15
LOAD RUN, 2-15
LOG, 2-6

M

MAT Addition, 2-7
MAT CON, 2-7
MAT CONVERT, 2-7
MAT COPY, 2-7
MAT Equality, 2-7
MAT IDN, 2-7
MAT INPUT, 2-7
MAT INV, 2-7
MAT MERGE, 2-7
MAT MOVE, 2-7
MAT PRINT, 2-7
MAT READ, 2-7
MAT REDIM, 2-7
MAT SEARCH, 2-8
MAT SORT, 2-8
MAT Subtraction, 2-8
MAT TRN, 2-8
MAT ZER, 2-8
MAX, 2-8
Microcommand sequence,
1-4 to 1-7
MIN, 2-8
MOD, 2-8
MOVE, 2-15
MOVE END, 2-15
MOVE TO, 2-15

N

NEXT, 2-8
NUM, 2-8

O

ON ERROR GOTO, 2-8
ON ERROR...GOTO, 2-3
ON/SELECT, 2-8

P

PACK, 2-9
PLOT, 2-9
POS, 2-9
PRINT, 2-9
PRINT AT, 2-9
PRINT BOX, 2-9
PRINT HEXOF, 2-5, 2-9
PRINT TAB, 2-9
PRINTUSING, 2-6, 2-9
PRINTUSING TO, 2-9

R

READ, 2-9
Register, 1-4, 1-8
Register-variable, 1-8
REM, 2-9
RENAME, 2-2, 2-10
RENUMBER, 2-12
RESET, 1-7, 1-8
RESTORE, 2-10
RETURN, 2-10
RETURN CLEAR, 2-10
REWIND, 2-16
RND, 2-10

ROTATE, 2-10
ROUND, 2-10
RUN, 2-12, 2-15

S

SAVE DA, 2-2, 2-16
SAVE DC, 2-2, 2-14
Scrambled Programs, 2-2
SCRATCH, 2-16
SCRATCH DISK, 2-16
SELECT, 1-2, 1-3
SELECT ERROR, 2-10
SELECT LINE, 2-10
SELECT ON/OFF, 2-2
SELECT TC, 2-2
SELECT [NO] ROUND, 2-10
SGN, 2-10
SKIP, 2-16
SOUND, 2-2, 2-10
SPACE, 2-10
Special Termination Character,
1-10, 1-12
SQR, 2-10
STEP, 2-11
STOP, 2-10, 2-11
STR, 1-11 to 1-13
SUB, 2-11, 2-12

T

TIME, 2-11
TRACE, 2-12
TRACE DISK, 2-12
Trig Functions, 2-11

U

UNPACK, 2-11

V

VAL, 1-11
Variable Name Length, 2-1
VER, 2-11
VERIFY, 1-5, 2-16

W

WRITE, 1-4

#

#ID FUNCTION, 2-17
#PART, 2-2, 2-5
#PI, 2-9
#TERM, 2-17

$

$ALERT, 2-2
$BREAK, 2-17
$CLOSE, 2-2, 2-17
$DISCONNECT, 2-2
$FORMAT, 2-2, 2-5,
$FORMAT DISK, 2-2, 2-15
$GIO, 1-4 to 1-9
$IF OFF, 1-2, 2-5
$IF ON, 1-3, 2-2, 2-5
$INIT, 2-6, 2-17
$MSG, 2-2, 2-17
$OPEN, 2-2, 2-17
$PACK, 2-9
$RELEASE PART, 2-2, 2-17
$RELEASE TERMINAL, 2-2, 2-17
$TRAN, 2-11
$UNPACK, 2-11

WANG

**Title** PC/2200 BASIC-2 COMPATIBILITY GUIDE

**Publications Number** 715-0005

# Customer Comment Form

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.

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

| | VERY GOOD | 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? | ☐ | ☐ | ☐ | ☐ | ☐ |

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

_____

_____

_____

Do you have any other comments or suggestions? _____

_____

_____

_____

**Name** _____

**Company** _____

**Street** _____

**City** _____

**State/Country** _____

**Zip Code** _____ **Telephone** _____

**Thank you for your help.**

Printed in U.S.A. 14-3151 8-83-5C

WANG

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD
FIRST CLASS PERMIT NO. 16 LOWELL, MA

POSTAGE WILL BE PAID BY ADDRESSEE

**WANG LABORATORIES, INC.**
**TECHNICAL PUBLICATIONS**
**ONE INDUSTRIAL AVENUE**
**LOWELL, MASSACHUSETTS 01851**

CUT ALONG DOTTED LINE

**WANG LABORATORIES, INC.**
ONE INDUSTRIAL AVENUE, LOWELL, MA 01851
TEL: 617/459-5000, TWX 710-343-6769, TELEX 94-7421


Printed in U.S.A.
715-0005
5-84