LC2/LC4

BASIC

Version 2.0/3.0

Form 201.6 July 1994

OPTO 22

43044 Business Park Drive • Temecula, CA 92590-3614
Phone: 800/321-OPTO (6786) or 909/695-3000
Fax: 800/832-OPTO (6786) or 909/695-2712
Internet Web site: http://www.opto22.com

Product Support Services:
800/TEK-OPTO (835-6786) or 909/695-3080
Fax: 909/695-3017
E-mail: support@opto22.com
Bulletin Board System (BBS): 909/695-1367
FTP site: ftp.opto22.com

This technical document describes the features, specifications, and operations of the product.

The information in this manual has been carefully checked and is believed to be accurate; however, no responsibility is assumed for possible inaccuracies or omissions. Specifications are subject to change without notice.

Opto 22 warrants all its products to be free from defects in material or workmanship for 24 months from the manufacturing date code.

This warranty is limited to the original cost of the unit only and does not cover installation, labor, or any other contingent costs.

Table of Contents

GENERAL INFORMATION

QUICK REFERENCE

LC2/LC4 COMMANDS, STATEMENTS and FUNCTIONS

APPENDICES

GENERAL INFORMATION

Important Things to Know

*** * * * READ THIS FIRST * * * ***

For those who are familiar with BASIC and want to begin using LC2/LC4 right away (and not read the manual), the following tips may save you hours of aggravation.

- Always use the NEW command before entering or downloading a program.
- When downloading a program written with IBM/Microsoft BASIC Interpreter, make sure it is in the ASCII format by using the **SAVE "FILENAME",A** command. Only ASCII files can be downloaded to the LC2/LC4.
- Make sure your programs do not have multiple statements on one line. The following line will generate a syntax error when LC2/LC4 tries to run it:

```
100     A=1 : B=2 : FOR I% = 1 TO 10 : NEXT
```

- Make a backup copy of the LC2/LC4 Utilities disk (Part Number 9970 or 8870) before using it. Be careful when using the upload command of LCTERM because any existing file with the specified file name will be overwritten when a file is uploaded.
- The Appendices contain valuable information. Please read them first.

Introduction

The LC2/LC4 Local Controller contains a BASIC Interpreter with a command set very similar to the IBM/Microsoft BASIC command set found on the IBM PC, XT, AT, PS/2, and compatibles. Since the LC2 and LC4 are local controllers without the peripherals associated with personal computers, (floppy disks, hard disks, color or monochrome displays and keyboards), many commands and statements used to communicate with these devices have been left out of the LC2/LC4 BASIC. To make efficient use of program space and to optimize execution speed, LC2/LC4 BASIC is limited to single-dimensional array variables with the maximum number of array elements equal to 255 and real-type variables limited to single precision. String and integer-type variables are supported.

The OPTOWARE driver is included in the system ROM in order to facilitate communications on the OPTOMUX network.

In addition to the OPTOWARE driver, a driver for the PAMUX family of I/O systems is also included in the LC4 BASIC ROM. The EX1 PAMUX bus daugther card is required to use the PAMUX driver.

*** * * IMPORTANT * * * IMPORTANT * * * IMPORTANT * * ***

Before any program is entered or downloaded, the NEW command must be issued. This insures that LC2/LC4 BASIC's pointers are reset to the beginning of the file. If the NEW command is not issued prior to entering or downloading a program, erroneous results may occur.

Calling the ROM-Based OPTOWARE Driver

The OPTOWARE driver allows a user to communicate with an OPTOMUX network by using the CALL statement. The LC2/LC4 OPTOWARE driver is very similar to the disk-based OPTOWARE driver for the IBM PC with one important exception. The LC2/LC4's driver is at a fixed location in memory and does not have to be loaded from a disk. Consequently, when assigning the offset location to the OPTOWARE variable, the ROM address of the driver must be used.

When using the IBM PC disk-based version of OPTOWARE, the following statements were used to load and declare the location of the driver:

```
10      DEF SEG = &H3300                'Define Segment For Loading Driver
20      BLOAD "DRIVER.COM",0            'Load Driver From Disk At Offset 0
30      OPTOWARE = 0                    'Driver Offset
```

These statements should be replaced with the following statement:

```
10      OPTOWARE = 4                    'Location Of ROM-based Driver
```

Parameters that are to be passed to the driver should be dimensioned the same way as shown in the OPTOWARE manual. Calling the driver is performed in the same manner as described in the disk-based OPTOWARE manual, Form 092.

The following is a simple example of using the LC2/LC4 ROM-based OPTOWARE driver to send a power up clear command to an OPTOMUX unit at address 255:

```
100     OPTOWARE = 4                    'Specify Offset Of Driver
120     DIM POSIT%(15)                  'Dimension POSITIONS To 16 Elements
130     DIM INFO%(15)                   'Dimension INFO To 16 Elements
140     DIM MODI%(1)                    'Dimension MODIFIERS To 2 elements
150     FOR I% = 0 TO 15
160     POSIT%(I%) = 0                  'Initialize POSITIONS Array To 0
170     INFO%(I%) = 0                   'Initialize INFO Array To 0
180     NEXT
190     MODI%(0) = 0                    'Initialize MODIFIERS To 0
200     MODI%(1) = 0
210     ERRCOD% = 0                     'Initialize ERRORS To 0
220     CMD% = 0                        'Set Power Up Clear Command
230     ADDR% = 255                     'Set OPTOMUX Address
.
.
300     '*
310     '*          Call The OPTOWARE Driver
320     '*
330     CALL OPTOWARE(ERRCOD%,ADDR%,CMD%,POSIT%(0), MODI%(0),INFO%(0))
340     IF ERRCOD% < 0 THEN PRINT "Return Error Code Is: ";ERRORS%
350     RETURN
```

LC2/LC4 BASIC Program Elements

An LC2/LC4 BASIC program is made up of a group of instructions which can perform a task or take some kind of action. More specifically, a program is composed of a number of lines. Each program line begins with a unique number followed by a command or statement. For the sake of simplicity and efficiency, LC2/LC4 BASIC is restricted to only one command or statement per line. The format of a typical line is as follows:

```
nnnnn    [COMMAND or STATEMENT] ['COMMENT]
```

Line Numbers:

"nnnnn" is the line number of the LC2/LC4 BASIC program line and must be an integer in the range of 0 to 32,767. Every line in an LC2/LC4 BASIC program must begin with a line number. Program lines are executed sequentially beginning at the lowest numbered line whenever a RUN command is given. If a BASIC statement is not preceded by a line number, LC2/LC4 BASIC will attempt to execute it directly.

Commands:

Commands are system level instructions such as LIST, RUN, and NEW which are most often used in direct mode to operate on an existing program or set up a new program. The LC2/LC4 BASIC commands may also be used within an LC2/LC4 program, however, it is up to the user to exercise care when using commands such as CLEAR or NEW in a program line.

Statements:

The LC2/LC4 BASIC interpreter is capable of executing a wide variety of statements. There are statements to perform input and output from the serial ports, statements to change the order of flow of program execution and statements that act on data items. Many statements include expressions.

Comment:

A comment is a non-executable statement used to explain a program's operation and/or purpose. It is regularly used as a form of documenting for a program. A comment is preceded by a single quote character (') or the REM statement.

LC2/LC4 BASIC Expressions

Expressions are an essential part of an LC2/LC4 BASIC program. They contain references to variables and constants (data items) and also to instructions which operate on or combine the data items to produce a desired result. The instructions that make up an expression are called operators. Expressions can be classified into four groups: numeric, string, relational, and functional, depending upon the type of results they yield.

Numeric Expressions

Numeric expressions are used to operate on numeric data using numeric operators. There are two types of numeric expressions: integer and real.

Integer: Those expressions involving integer variables or constants. An integer is defined as a whole number in the range of -32,767 to +32,767 (16 bits).

Real: An expression which contains at least one term of the type REAL. A REAL may have a fractional part or be a number which falls outside the integer range. The range for real numbers used by LC2/LC4 BASIC is from ±1.1175E-38 to 3.40E+38. Real numbers are sometimes referred to as floating point numbers.

String Expressions:

String expressions are those that are used to operate on or combine string (text) variables or constants. A string data item or the result of a string expression is limited to 255 characters.

Relational Expressions:

Relational expressions are used to compare data items and provide a result which is either true or false. This result is always an integer number with a value of -1 if the result is true or a value of 0 if the result is false. Relational expressions can operate on numeric or string data by testing for conditions using the six different relational operators. Relational operators are useful for making decisions regarding program flow, and may be combined within an expression using logical operators.

Functional Expressions:

Functional expressions are used to execute predetermined operations on an operand.

LC2/LC4 BASIC Operators

Arithmetic Operators:

The following arithmetic operators are listed in order of precedence:

Operator	Operation	Sample Expression
-	Negation	-A
*	Multiplication	A * B
/	Floating Point Division	A / B
\	Integer Division	A \ B
MOD	Modulo Arithmetic	A MOD B
+	Addition	A + B
-	Subtraction	A - B

Integer division using the \ operator will round the two operands to integers, perform the division and then truncate the result to an integer. For example:

```
10      X = 22.57\5.99
20      PRINT X
RUN
3

ok
```

The MOD operator performs the modulo function which returns the integer value of the remainder after performing an integer division. For example:

```
10      X = 8 MOD 3
20      PRINT X
RUN
2

ok
```

Logical Operators:

Logical operators perform Boolean or logical operations on numeric values and are useful for joining relational expressions. Operations using the logical operators return a true or false value. Operands are considered true if they are not equal to zero and false if they are equal to zero. A logical operation is performed on a bit-by-bit basis. The following is a list of logical operators used by LC2/LC4 BASIC.

NOT

A	NOT A
F	T
T	F

AND

A	B	A AND B
F	F	F
F	T	F
T	F	F
T	T	T

OR

A	B	A OR B
F	F	F
F	T	T
T	F	T
T	T	T

XOR

A	B	A XOR B
F	F	F
F	T	T
T	F	T
T	T	F

The order of precedence of logical operators is NOT, AND, OR, and XOR.

For example, two numbers, 28 and 36 (011100, 100100 in binary), are used in the following examples to illustrate logical operations:

28 AND 36 = 4	Bit by Bit:	28 = 011100 36 = 100100 4 = 000100
28 OR 36 = 60	Bit by Bit:	28 = 011100 36 = 100100 60 = 111100
28 XOR 36 = 56	Bit by Bit:	28 = 011100 36 = 100100 56 = 111000

Relational Operators:

Relational operators are used to compare two values and return either a true (-1) or a false (0) result. Program flow can then be altered based on this result. Relational operators will work with either numeric operands or string operands. However, both operands in an expression must be of the same type. The following is a table of relational operators and their functions:

Operators	Relation	Example
=	Equality	A = B
<> or ><	Inequality	A <> B
<	Less than	A < B
>	Greater than	A > B
<= or =<	Less than or equal to	A <= B
>= or =>	Greater than or equal to	A >= B

Note On String Comparisons:

String comparisons are performed alphabetically by a character's ASCII value. Each string operand is compared one character at a time starting with the first (left-most) character of each string. A character is considered less than another character if the ASCII value of the former is less than the ASCII value of the latter. A string with fewer characters than another string is always less than the longer string. Leading and trailing blanks are also evaluated and are therefore significant. When string constants are used they must be enclosed in quotation marks. The following examples are of relational expressions which are all true:

```
5 = 5
47 <= 47
47 <= 65
"XX" < "xx"
"x" < "x "
"230" < "470"
"myname" > "Myname"
10 <> 15
```

Functional Operators:

Functional operators are used to perform a predetermined operation on an operand. Functional operators are grouped into two categories: numeric and string functions. Number functions return a numeric value and string functions return a string value. The following are examples of functional operators:

ABS(x)	Returns the absolute value of x
COS(x)	Returns the cosine of angle x (in radians)
DATE$	Returns the system date
LEN(x$)	Returns the number of characters in x$

QUICK REFERENCE

List of Reserved Words

There are many words which have a special meaning to the LC2/LC4 BASIC. These words are defined as "Reserved Words" and include all the operators, commands, functions, and statements of LC2/LC4 BASIC. Reserved words cannot be used as variable names. In order for LC2/LC4 BASIC to understand the set of operators, commands, statements, and functions, these words must be appropriately delimited. Proper delimiting is accomplished by using spaces (or other characters allowed by the syntax of each command) to separate the reserved words from data or other reserved words.

LC2 Release 2 Reserved Words

ABS	DATE$	HEX$	LLIST	POKE	THEN
ADDRESS@	DELAY	HIDE	LOC	PRINT	TIME$
AND	DELETE	IF	LOF	READ	TIMER
AS	DIM	INKEY$	LOG	REM	TO
ASC	ELSE	INP	LPRINT	RESTORE	TROFF
AWAKE	END	INPUT	MID$	RESUME	TRON
CALL	ERL	INPUT$	NEW	RETURN	VAL
CHR$	ERR	INSTR	NEXT	RIGHT$	VARPTR
CINT	ERROR	INT	NOT	RUN	WEND
CLEAR	EXP	KEY	OFF	SIN	WHILE
CLOSE	FIX	LEFT$	ON	SLEEP	XOR
COM	FOR	LEN	OPEN	SQR	
CONT	FRE	LET	OR	STEP	
COS	GOSUB	LINE	OUT	STOP	
DATA	GOTO	LIST	PEEK	STR$	

LC4 Release 3 Reserved Words

ABS	DATE$	GOTO	LIST	PEEK	THEN
ADDRESS@	DELAY	HEX$	LLIST	POKE	TIME$
AND	DELETE	HIDE	LOC	PRINT	TIMER
APPEND	DIM	IF	LOF	READ	TO
AS	ELSE	INKEY$	LOG	REM	TROFF
ASC	END	INP	LPRINT	RESTORE	TRON
AWAKE	EOF	INPUT	MID$	RESUME	VAL
CALL	ERL	INPUT$	NEW	RETURN	VARPTR
CHR$	ERR	INSTR	NEXT	RIGHT$	WAIT
CINT	ERROR	INT	NOT	RUN	WEND
CLEAR	EXP	KEY	OFF	SIN	WHILE
CLOSE	FILES	KILL	ON	SLEEP	XOR
COM	FIX	LEFT$	OPEN	SQR	
CONT	FOR	LEN	OR	STEP	
COS	FRE	LET	OUT	STOP	
DATA	GOSUB	LINE	OUTPUT	STR$	

Commands

The following is a list of all the commands included in LC2/LC4 BASIC. More detailed information, including examples, can be found in the "LC2/LC4 Commands, Statements, And Functions" section.

COMMAND	ACTION
ADDRESS@	Variable used for assigning a unique address to a LC2/LC4 for selective host to LC2/LC4 communications.
AWAKE address	Enables multidropped LC2/LC4's for selective host LC2/LC4 communications.
CLEAR	Clears all variables in program and allocates space for machine language programs.
CONT	Continues program execution.
DELETE line1-line2	Deletes specified lines from program.
FILES	Displays the names of files in RAM disk (LC4 only).
HIDE	Prevents a program from being listed.
KILL filespec	Delete a file from the RAM disk (LC4 only).
LIST	Lists entire program.
LIST line1-line2	Lists program lines specified.
LLIST line1-line2	Lists program lines out the OPTOMUX communications port.
NEW	Erases the current program in memory and its associated variables.
RUN	Executes a program, beginning at the first program line.
SLEEP address	Disables multidropped LC2/LC4's for selective host to LC2/LC4 communications.
TRON, TROFF	Turns on or off the trace feature.

Statements

This section provides a quick description of all the LC2/LC4 BASIC statements. The list tells what each statement does and shows the syntax. For the more complex statements, the syntax shown may not be complete. For a complete description, refer to the "LC2/LC4 Commands, Statements, And Functions" section.

STATEMENT	ACTION
CALL numvar (variable list)	Calls a machine language program.
CLOSE #f	Closes the logical device or file.
DATA list of constants	Creates a list of data to be accessed by the READ statement.
DATES = x$	Used to set the system date.
DELAY n	Causes BASIC to wait for a time specified by n.
DIM list of subscripted variables	Declares maximum subscript values for arrays and allocates space for them.
END	Stops the program, closes all files, and returns to command level.
FOR variable = x TO y STEP z	Repeats program lines a number of times. The NEXT statement closes the loop.
GOSUB line	Calls a subroutine by branching to the specified line. The RETURN statement returns from the subroutine.
GOTO line	Branches to the specified line.
IF expression THEN statement ELSE statement	Performs the statement following the THEN if the expression is true (non-zero). Otherwise, performs the statement following the ELSE or goes to the next line.
INPUT "prompt";variable list	Reads data from the host communications port.
LET v=expression	Assigns the value of the expression to the variable v.
LINE INPUT "prompt";stringvar	Reads an entire line from the host input port, ignoring commas or other delimiters.
LPRINT list of expressions	Prints data on a printer connected to the OPTOMUX communications port.
NEXT v	Closes a FOR... NEXT loop (see FOR).
ON KEY(x$) GOSUB line	Branches to subroutine at line when a trapped character is received at host port (COM0).
ON TIMER(n) GOSUB line	Branches to subroutine at line when n seconds have elapsed.

Statements (continued)

STATEMENT	ACTION
ON COM(n) GOSUB line	Branches to a subroutine when a character is received on the specified serial port.
ON ERROR GOTO line	Branches to a subroutine when a BASIC error occurs.
ON n GOSUB line list	Branches to subroutine at list item n.
ON n GOTO line list	Branches to statement at list item n.
OPEN filespec	Selects or creates a file in the RAM disk (LC4 only).
OPEN "COM... " AS #f	Opens a communications port and assigns it to logical file f.
OUT m,n	Outputs the byte n to a LC2/LC4 I/O location m.
POKE m,n	Puts byte n into memory at location m.
PRINT list of expressions	Writes the list of expressions to the host communications port.
READ variable list	Retrieves information from the data list set up by the DATA statement.
REM remark	Includes remark in program.
RESTORE line	Resets the DATA pointer to beginning of list.
RESUME	Continues program execution after an error recovery routine is performed.
RETURN	Returns from subroutine.
STOP	Stops program execution, prints a break message, and returns to command level.
TIME$ = x$	Used to set the system time.
TIMER	Returns a number indicating number of seconds since midnight.
WAIT	Suspends execution of a program while monitoring the status of an I/O port (LC4 only).
WEND	Closes a WHILE... WEND loop.
WHILE	Statements within a loop will execute while expression remains true.

Functions

The LC2/LC4 BASIC contains many built-in functions which are listed below. These functions can be grouped into two separate categories: numeric functions and string functions. Numeric functions return a numeric result and string functions return a string result. For a complete description, refer to the "LC2/LC4 Commands, Statements, And Functions" section.

Numeric Functions

FUNCTIONS	RESULT
ABS(x)	Returns the absolute value of x.
ASC(x$)	Will return the ASCII code of the first character in x$.
CINT(x)	x is converted to an integer by rounding.
COS(x)	Returns the cosine of angle x (in radians).
EOF(filenum)	Indicates an end of file condition (LC4 only).
ERL	Returns the line number associated with an error.
ERR	Returns the error number associated with an error.
EXP(n)	Calculates the exponential value of an expression.
FIX(x)	Truncates x to an integer.
FRE(x)	Returns the amount of free space in memory not currently in use by BASIC.
INP(x)	Reads a byte from LC2/LC4 I/O location x.
INT(x)	Returns the largest integer which is less than or equal to x.
LOC(f)	Returns the number of characters in the specified communications buffer.
LOF(f)	Returns the number of free bytes in the specified communications buffer.
LOG(n)	Calculates the natural log of a numeric expression.
PEEK(m)	Reads the byte in memory location m.
SIN(x)	Returns the sine of angle x (in radians).
SQR(x)	Returns the square root of x.
VAL(x$)	Returns the numeric value of the string x.
VARPTR(variable)	Returns the address in memory of variable.

String Functions

FUNCTIONS	RESULT
CHR$(x)	Returns the character with ASCII code x.
DATE$	Returns the system date.
HEX$(n)	Returns a string representing the hexadecimal value of the decimal argument.
INKEY$	Reads a character from the host serial communications port.
INPUT$(n,#f)	Reads n characters from the specified logical device or file.
INSTR(n,x$,y$)	Searches for a substring within a string.
LEFT$(x$,n)	Returns the left-most n characters of x$.
LEN(x$)	Returns the number of characters in x$.
MID$(x$,m,n)	Returns the requested part of a specified string.
RIGHT$(x$,n)	Returns the right-most n characters of x$.
STR$(x)	Converts x to a string value.
TIME$	Returns the time from the LC2/LC4 real-time clock.

COMMANDS, STATEMENTS, & FUNCTIONS

ABS Function

Purpose:

Returns the absolute value of an expression.

Version:

LC2, LC4

Format:

v=ABS(x)

Comments:

x may be any numeric expression. The absolute value of an expression will always evaluate to a positive number or zero.

Example:

```
PRINT ABS (3 * (-2))
6

ok
```

ADDRESS@ Variable

Purpose:

To assign an unique address to an LC2/LC4 local controller. Used in conjunction with the SLEEP and AWAKE commands when several LC2/LC4's are multidropped on the same communications link.

Version:

LC2, LC4

Format:

ADDRESS@ = x

x is an integer value.

Comments:

The ADDRESS@ variable is not necessary unless more than one LC2/LC4 will be connected to the same host communication lines and the SLEEP and AWAKE commands will be used to enable or disable individual LC2/LC4 local controllers. The ADDRESS@ = x statement should then be issued from the host terminal to only one LC2/LC4 local controller at a time. All other LC2/LC4's should be powered down when an address is being assigned.

The ADDRESS@ variable is a special system variable which cannot be cleared by a CLEAR or NEW command. The only way to change this variable is to reassign it using the ADDRESS@ = x statement.

NOTE: The LC4 has hardware address jumpers to set the ADDRESS@ value. Therefore, the ADDRESS@ = x statement is not valid; however, the PRINT ADDRESS@ statement can return the LC4 address. Please refer to the LC4 Hardware Description And Installation manual (Form 157) on setting up the address jumpers.

Also see the AWAKE and SLEEP commands.

Example:

See the APPENDICES section on Multidropping Several LC2/LC4's.

ASC Function

Purpose:

Returns the corresponding ASCII code for the first character of the string x$.

Version:

LC2, LC4

Format:

v = ASC(x$)

x$ may be any string expression.

Comments:

The ASC function will return a numerical value that is the ASCII code of the first character of he string x$. If the string x$ is null, an "illegal function call" error will be returned.

The CHR$ function is the complement of the ASC function, and it converts the ASCII code to a character. Refer to the APPENDICES for a list of ASCII codes and their corresponding characters.

Example:

```
100     O$ = "OPTOMUX"
110     PRINT ASC(O$)
RUN
79

ok
```

This example shows that the ASCII code for a capital O is 79. PRINT ASC ("OPTOMUX") would also give the same result.

AWAKE Command

Purpose:

To enable a single LC2/LC4 or all LC2/LC4's on a multidropped, communications line in order to download programs to selected units.

Version:

LC2, LC4

Format:

AWAKE [x]

x is an integer value which indicates the address of the LC2/LC4 unit to enable.

Comments:

If no address is specified when the AWAKE command is issued, all LC2/LC4 units on the same communications link which are powered up and in the sleep mode will be enabled.

The AWAKE command is normally issued from the host terminal or computer when different programs must be downloaded to separate LC2/LC4's. Individual units can be made active using the AWAKE command and a specified address which will match the previously configured ADDRESS@ variable of the selected LC2/LC4.

Also see the ADDRESS@ variable and SLEEP command.

Example:

See the APPENDICES section on Multidropping Several LC2/LC4's.

CALL Statement

Purpose:

Calls a machine language subroutine.

Version:

LC2, LC4

Format:

CALL numvar [(variable [,variable,...])]

numvar is the name of a numeric variable. The value of the variable indicates the starting memory address of the subroutine being called.

variable is the name of a variable which is to be passed as an argument to the machine language subroutine.

Comments:

The CALL statement is a way of interfacing machine language programs with BASIC. Even if no variables are to be passed, a dummy variable must be specified in order for the CALL statement to work.

Example:

```
100     JOE = 34210
110     CALL JOE(A,B$,C)
```

The variables A, B$, and C are passed as arguments to the machine language subroutine which begins at location 34210 in memory. Calling the OPTOWARE driver:

```
10      OPTOWARE = 4
.
.
.
320     CALL OPTOWARE(ERRORS%,ADDRESS%,COMMAND%,POSITIONS%(0),MODIFIERS%(0),INFO%(0))
```

The variables ERRORS%, ADDRESS%, COMMAND%, POSITIONS%(0), MODIFIERS%(0), and INFO%(0) are passed as arguments to the OPTOWARE driver.

The OPTOWARE driver is a machine language program which exists in the LC2/LC4 System ROM.

CHR$ Function

Purpose:

Converts an ASCII code to its corresponding character.

Version:

LC2, LC4

Format:

v$ = CHR$(x)

Comments:

The CHR$ function returns an one character string with the corresponding ASCII code x. The complementary function of CHR$( ) is ASC( ), which converts a character back to its corresponding ASCII code. (LC2/LC4 BASIC only uses 7 bit ASCII, the ASCII characters from 0 to 127.) Refer to the APPENDICES for a list of ASCII codes and their corresponding characters.

CHR$ can be used to send both printable and non-printable characters to devices such as terminals or OPTOMUX stations. For example, the BEL character (CHR$(7)) can be sent to the host to beep the speaker to get the operator's attention.

Example:

```
PRINT  CHR$(89)+CHR$(101)+CHR$(115)
Yes

ok
```

CINT Function

Purpose:

Converts a number or expression to an integer by rounding.

Version:

LC2, LC4

Format:

v = CINT(x)

x may be any numeric expression that is within the range of -32,767 to 32,767. If x is greater than 32,767 or less than -32,767, CINT will return an overflow error.

Comments:

The CINT function converts x to an integer by rounding the fractional portion.

The main differences between CINT, INT, and FIX are as follows: CNT rounds a number to the nearest integer, INT returns the largest integer less then or equal to the number, and FIX removes the fractional part of a number without rounding.

Example:

```
PRINT CINT(53.89)
54

ok
PRINT CINT (-1.66)
-2

ok
```

Rounding selects a higher number on positive numbers and selects a more negative number on a negative number.

CLEAR Command

Purpose:

Clears all variables by setting numeric variables to zero and string variables to null. An option Allows the beginning of the LC2/LC4 BASIC program space to be specified.

Version:

LC2, LC4

Format:

CLEAR [m]

m is a byte count used to set the amount of memory space to reserve between the start of RAM and the beginning of the BASIC program. The m option is necessary for storing machine language routines.

Comments:

The CLEAR command will clear all variables without erasing the current program in memory. Arrays will be undefined, strings will have a null value, and numeric values will be set to zero after a CLEAR command is issued.

Example:

The following example clears all variables without erasing the program in memory:

```
CLEAR
```

The next example clears all variables and allocates 2K bytes of memory at the beginning of the RAM space to store machine language routines or data. The POKE statement can then be used to store a machine language program or data into this reserved space.

```
CLEAR 2048
```

CLOSE Statement

Purpose:

Ends the association between a communications port or a file and the logical file number assigned to it.

Version:

LC2, LC4

Format:

CLOSE [[#]filenum]

filenum is the logical file number used on the OPEN statement.

Comments:

After the CLOSE command is executed, any subsequent references to that logical file number will be invalid. The execution of a RUN, NEW, or END statement will automatically close any previously executed OPEN "COM... " statement. When the CLOSE statement is executed, any characters remaining in the input buffer will be lost and the buffer pointers will be reset.

Example:

This example ends the association of logical file #2 and the communications port. It is assumed that this association was previously set by the OPEN "COM... " AS #2 statement.

```
100     CLOSE #2
```

CONT Command

Purpose:

Continues execution of a program after a break.

Version:

LC2, LC4

Format:

CONT

Comments:

The CONT command can be used to resume execution of a program after a Ctrl-C has been pressed, or a STOP statement has been executed. Execution continues at the point where the program was interrupted. If the program was interrupted after a prompt from an INPUT statement, execution continues with the reprinting of the prompt.

CONT can be very useful when used in conjunction with STOP for debugging purposes. When execution is stopped, the values of variables can be examined or changed using direct mode statements. CONT is then used to resume execution. CONT is invalid if program lines have been added, deleted, or modified after the break.

Example:

The following example contains several STOP statements which interrupt program execution:

```
100     A = 30
110     PRINT A
120     A = A+10
130     STOP
140     A = A+20
150     STOP
160     A = A + 50
170     PRINT A
180     STOP
RUN


30
break in line 130
ok      (At this point, we issue the direct commands PRINT A and CONT)
PRINT A
40

CONT
break in line 150
ok      (At this point, we again issue the direct commands PRINT A and CONT)
PRINT A
60

CONT
110
break in line 180

ok
```

COS Function

Purpose:

Returns the cosine of the argument value which is expressed in radians.

Version:

LC2, LC4

Format:

v = COS(x)

x is the angle in radians whose cosine is to be calculated.

Comments:

The result returned by the COS function is a single precision, real value. To convert a value in degrees to radians, multiply the number of degrees by 0.01745329.

Example:

This example shows, first, that the cosine of 90 degrees is equal to 2.808803E-07 (very close to zero). Then it calculates the cosine of 90*2 or 180 degrees and the result is -1.

```
100     DEG2RAD = 0.01745329                'Obtained From 3.141593/180
110     DEGREES = 90
120     RADIANS = DEGREES * DEG2RAD
130     PRINT COS (RADIANS)
140     PRINT COS (RADIANS*2)
RUN
2.808803E-07
-1

ok
```

DATA Statement

Purpose:

Stores a list of numeric and string constants that are to be accessed by the program's READ statement(s).

Version:

LC2, LC4

Format:

DATA constant [,constant,...]

constant may be a numeric or string constant. Expressions are not allowed in the list. Numeric constants may be in integer, floating point, fixed point, or hex format. String constants need only be enclosed in quotation marks when the string must contain commas, significant leading, or trailing blanks.

Comments:

DATA statements may be placed anywhere in the program, and may contain as many constants as will fit on a line and a program may contain any number of DATA statements. The lists created by several DATA statements can be considered one long sequential list of information that is accessed by the READ statement in line number order. A RESTORE command can be used to reset the current DATA line pointer to a particular line or to the beginning of the program. Different constant types can be contained in the DATA statement but they must match the corresponding variable type in the READ statement or an error will occur. DATA statements are non-executable statements.

Example:

See examples under the READ Statement.

DATE$ Variable And Statement

Purpose:

Used to set or read the system date.

Version:

LC2, LC4

Format:

As a variable: v$ = DATE$

As a statement: DATE$=x$

Comments:

For the variable v$ = DATE$:

When using DATE$ as a variable, a 10 character string of the form mm-dd-yyyy is returned. The two digits of the month are represented by mm, the day is represented by the two digits dd and the year is represented by the four digits yyyy.

For the statement DATE$ = x$:

The string expression x$ is used to set the current date. This expression can be specified in one of the following ways:

mm-dd-yy
mm/dd/yy
mm-dd-yyyy
mm/dd/yyyy

If only one digit is given, a zero is assumed for the first digit in the month, day, or year. If only two digits are specified for the year, the year is assumed to be 19yy.

Example:

The following example sets the date (in this case September 22, 1987), then reads it back and displays it. Notice how LC2/LC4 BASIC inserted a leading zero for the month and assumed the first two digits of the year were 19.

```
10      DATE$ = "9/22/87"
20      PRINT  DATE$
RUN
09-22-1987

ok
```

DELAY Statement

Purpose:

Cause the BASIC to wait for a specified time.

Version:

LC2, LC4

Format:

DELAY x

x is a numeric expresion in the range 0.1 to 3276.7 which represents time in seconds for LC2. The range of x for LC4 is from 0.01 to 42,949,669.99 seconds. A fractional value will specify fractions of a second down to a resolution of 100 milliseconds for the LC2 and 10 milliseconds for the LC4. A value entered that is outside this range results in an "overflow" error.

Example:

```
100     DELAY 5.7     'Wait For 5.7 Seconds
```

DELETE Command

Purpose:

Deletes program lines.

Version:

LC2, LC4

Format:

DELETE [line1] [-line2]

line1, line2 are valid line numbers in the range 0 to 32,767. line1 is the first or only line to be deleted. line2 is the last line to be deleted.

Comments:

DELETE will erase all the program lines specified in the range. LC2/LC4 BASIC will always return to the command level after the DELETE command has been executed. If a line number is specified which does not exist, an "illegal function call" error will occur.

Example:

This example deletes line 640:

```
DELETE 640
```

The next example will erase lines 50 through 200, inclusive:

```
DELETE 50 - 200
```

This last example shows how all the lines up to and including, line 137 can be erased:

```
DELETE -137
```

DIM Statement

Purpose:

Allocates storage space in memory for array variables and specifies the maximum values for array variable subscripts.

Version:

LC2, LC4

Format:

DIM variable(subscript) [,variable(subscript),...]

variable is the name to be used for the array.

subscript is a numeric expression defining the maximum subscript of the array (0-255).

Comments:

When the DIM statement is executed, all the elements of the specified, numeric arrays are set to an initial value of zero. String array elements are all set to an initial null value (zero length).

Array variable names used without a DIM statement are assigned a maximum subscript value of 10. Using a subscript that is greater than the maximum specified, results in an error.

The minimum value for a subscript is always 0. Arrays are limited to one dimension.

If you try dimensioning an array more than once, an error will occur.

Example:

The following example shows the integer array variable D%, the real array variable E, and the string array variable F$ being dimensioned to all contain 21 elements.

```
100     DIM D%(20), E(20), F$(20)
```

END Statement

Purpose:

Terminates execution of a program and returns to the command level.

Version:

LC2, LC4

Format:

END

Comments:

END statements may be placed anywhere in the program to terminate execution and a program may contain multiple END statements. The END statement does not print the "break in line" message as does the STOP statement. LC2/LC4 BASIC will always return to the command level after an END is executed. The END statement at the end of a program is optional.

Example:

This example ends the program if K is less than 500; otherwise, the program will branch to line number 100.

```
900     IF K < 500 THEN END ELSE GOTO 100
```

EOF Function

Purpose:

Indicates an End Of File condition.

Version:

LC4

Format:

v = EOF(filenum)

Comments:

filenum is the number that was specified in the OPEN filenum statement.

The EOF function returns a true (-1) if the end of the file has been reached or a false (0) if the end of the file has not been reached. By using this function you can avoid an "input past end" error.

The EOF function can also be used with a communications port that has been opened as a logical file. A -1 is returned if the communications buffer is empty.

Example:

This example reads information from a sequential file named "ALARMS" residing in the RAM disk area of the LC4. Values are read into the string array ALARM$ until the end of file is reached.

```
100     OPEN "ALARMS" FOR INPUT AS #1
110     NDX% = 0
120     IF EOF(1) THEN END
130     INPUT #1, ALARM$(NDX%)
140     NDX% = NDX% + 1
150     GOTO 120
```

ERL Variable

Purpose:

Returns the line number associated with an error.

Version:

LC2, LC4

Format:

v = ERL

Comments:

If there have been no errors since the LC2/LC4 has been powered up, ERL will contain a zero, otherwise ERL will contain the line number of the last error detected.

Example:

```
100     ON ERROR GOTO 1000
.
.
.
150     PRINT EXP(X)
.
.
.
1000    IF ERL <> 50 THEN GOTO 2000          'Check For Error Line
1010    X = 88.72283                         'Set Maximum Value
1020    RESUME NEXT                          'And Resume On Next Line
```

ERR Variable

Purpose:

Returns the error number associated with an error.

Version:

LC2, LC4

Format:

V = ERR

Comments:

If there have been no errors since the LC2/LC4 has been powered up, ERR will contain a zero, else ERR will contain an error value. In direct mode, ERR will always contain a zero. Refer to the APPENDICES for a complete list of error messages.

Example:

```
100     ON ERROR GOTO 1000
.
.
.
150     Z = EXP(X)
.
.
.
1000    IF ERR<>6 THEN GOTO 2000            'Check For Overflow
1010    IF ERL<>150 THEN GOTO 2000          'Check Line Number
1020    Z = 88.72283                        'Set Maximum Value
1030    RESUME NEXT                         'And Resume On Next Line
```

EXP Function

Purpose:

Calculate the exponential value of the specified expression.

Version:

LC2, LC4

Format:

v = EXP(x)

x = can be any numeric expression between -88.33654 to 88.72283.

Example:

Calculate e raised to the 23.4 power:

```
100     X = EXP(23.4)
```

FILES Command

Purpose:

Displays the names of the files residing in the RAM disk area of the LC4.

Version:

LC4

Format:

FILES

FIX Function

Purpose:

Truncates x to an integer.

Version:

LC2, LC4

Format:

v% = FIX(x)

x may be any numeric expression.

Comments:

The FIX function returns only the digits to the left of the decimal point. The major difference between the FIX and the INT functions is that FIX does not return the more negative number when x is negative.

The main differences between CINT, INT, and FIX are as follows: CNT rounds a number to the nearest integer, INT returns the largest integer less than or equal to the number, and FIX removes the fractional part of a number without rounding.

Example:

```
PRINT FIX(53.89)
53

ok
PRINT FIX(-1.66)
 -1

ok
```

FOR And NEXT Statements

Purpose:

Executes the set of instructions within the loop a specified number of times.

Version:

LC2, LC4

Format:

FOR variable = x TO y [STEP z]
.
.
.
NEXT [variable][,variable,...]

variable is an integer or single precision variable used as a counter or index for the loop.

x is the initial value of the counter. It may be a numeric expression.

y is the final value of the counter. It may be a numeric expression.

z is the increment value for the counter. It may be a numeric expression.

Comments:

Program lines which follow the FOR statement are executed until the NEXT statement is encountered. Then the counter is incremented by the amount specified by the STEP value z. If the STEP value is not specified, a value for z is assumed to be 1. A check is then performed to see if the value of the counter is now greater than the final value y. If it is not greater, BASIC branches back to the first line after the FOR statement and the process is repeated. If it is greater, execution will continue with the statement following the NEXT statement.

If the initial value of x is greater than the final value y, then a negative step value for z must be given. If a negative step value is not given, the FOR statement will not evaluate and execution will continue with the statement following the NEXT statement. If a negative step value for z is given, the counter is decremented each time through the loop, and the loop is executed until the counter is less than the final value.

If z is zero, an infinite loop will be created unless you provide some way to set the counter greater than the final value.

Program performance will be improved if you use integer counters whenever possible and no variable is referenced with the NEXT statement.

FOR And NEXT Statements (Continued)

Nested Loops

FOR... NEXT loops may be nested (one FOR... NEXT loop may be placed inside another FOR... NEXT loop). If loops are nested, each loop must have an unique variable name as its counter. The NEXT statement for the inside loop must appear before that for the outside loop. If nested loops have the same end point, a single NEXT statement may be used for all of them.

A NEXT statement of the form

NEXT var1, var2, var3

is equivalent to the sequence of statements:

NEXT var1
NEXT var2
NEXT var3

If the variable(s) in a NEXT statement are omitted, the NEXT statement will match the most recent FOR statement. If nested FOR... NEXT loops are used, the variable(s) on all the NEXT statements should be included, especially if there is any branching out of the loops. To avoid confusion and possible errors, all branches out of a loop should be done with GOSUB statements rather than GOTO statements. The GOSUB statement insures that execution will return to statements inside the loop so a proper exit out of the FOR... NEXT loop can take place. If a premature exit must be performed, just set the counter variable to a value larger than the specified final value y and branch to the NEXT statement (if STEP value is positive). For negative STEP values, set the counter variable to a smaller value than the specified final value.

If a NEXT statement is encountered before its corresponding FOR statement, a "NEXT without FOR" error occurs.

Example:

The following example shows how a FOR... NEXT loop is used to send a "Power Up Clear" and a "Reset" command to a group of OPTOMUX stations. There are 10 OPTOMUX stations with the following addresses: 2, 4, 6, 8, 10, 12, 14, 18, 20, and 22.

```
100     FOR ADDR% = 2 TO 22 STEP 2
110     CMD% = 0              'Power Up Clear Command
120     GOSUB 5000            'Call OPTOWARE Driver
130     CMD% = 1              'Reset Command
140     GOSUB 5000            'Call OPTOWARE Driver
150     NEXT
.
1000    END
5000    CALL OPTOWARE(ERRCOD%,ADDR%,CMD%,POSITIONS%(0),MODIFIERS%(0),INFO%(0))
5010    IF ERRCOD% < 0 THEN PRINT ERRCOD%; CMD%;ADDR%
5020    RETURN
```

FRE Function

Purpose:

Returns the number of bytes in memory that are not being used by BASIC.

Version:

LC2, LC4

Format:

v = FRE(x)

x is a dummy argument

Comments:

The FRE function will return the value which corresponds to the difference between the end of the program (including the variable storage area) and the LC2/LC4 BASIC's symbol table boundary. Since the variable storage area is allocated only after the RUN statement is executed, a FRE statement issued at the beginning of a program gives a more realistic value of free program space. Since string variables are dynamic, their initial allocation may not correspond to the amount of memory which may be used up as the strings grow throughout a program.

Example:

This example may not give a realistic value of free memory if a program currently in memory uses many variables:

```
PRINT FRE(0)
15752

ok
```

A more exact value can be found by inserting the FRE statement in the program, as follows:

```
100     '*    Insertion of FRE statement at beginning of program
200     '*
300     PRINT "THE AMOUNT OF FREE PROGRAM SPACE IS: ";FRE(0)
400     PRINT
500     STOP
600     '*    User Program Starts Here
700     PRINT "WELCOME TO MY PROGRAM"
.
.
.
1000    END
RUN
THE AMOUNT OF FREE PROGRAM SPACE IS: 11045
break in line 500


CONT
WELCOME TO MY PROGRAM
.
.
.
```

The actual value returned by FRE on your computer may differ from this example.

GOSUB And RETURN Statements

Purpose:

To branch to a subroutine and then return.

Version:

LC2, LC4

Format:

```
GOSUB line
.
.
.
RETURN
```

line is the number of the first line of the subroutine.

Comments:

Subroutines may be nested (called from within another subroutine) and a subroutine may be called any number of times. The depth to which subroutines are nested is limited only by the available memory.

Each subroutine must have at least one RETURN statement. Upon encountering a RETURN statement, LC2/LC4 BASIC will branch back to the instruction following the most recent GOSUB statement. Subroutines may appear anywhere in a program and may contain more than one RETURN statement.

If subroutines are placed in the middle of your program, the STOP, END, or GOTO statements may be used to direct program execution around the subroutine. Use the ON... GOSUB statement to branch to different subroutines based on the result of an expression.

Example:

This example shows how a subroutine works. The GOSUB in line 100 calls the subroutine in line 130. So the program branches to line 130 and starts executing statements there until it sees the RETURN statement in line 180. At that point, the program goes back to the statement after the subroutine call (it returns to line 110). The END statement in line 120 prevents the subroutine from being performed a second time.

```
100     GOSUB 130
110     PRINT "WE ARE BACK FROM THE SUBROUTINE"
120     END
130     '*    This is a subroutine
140     PRINT "THE SUBROUTINE"
150     PRINT "IS EXECUTING"
160     X = 100
170     PRINT X
180     RETURN
RUN
THE SUBROUTINE
IS EXECUTING
 100
WE ARE BACK FROM THE SUBROUTINE

ok
```

GOTO Statement

Purpose:

Unconditionally branch to the specified line number.

Version:

LC2, LC4

Format:

GOTO line

line is the line number (0 to 32,767) in the program.

Comments:

Upon execution of a GOTO statement, LC2/LC4 BASIC will transfer control to the statement at line. The statement apearing at line and those following are then executed. If a non-executable statement (such as REM or DATA) is encountered at line, the program continues at the first executable statement encountered after line. If the line number does not exist, an "unknown line" error will occur.

The GOTO statement can also be used at the command level to re-enter a program at a desired point.

Example:

This example illustrates a loop that increments the variable I% until the variable equals the value 100. Line 100 initializes I% to start at 1. Line 110 increments the variable I% and line 120 prints it. Line 130 is used to check if the value has reached 100, then branches to the end of the program. If the value is not 100, the program branches to line 110 and continues incrementing the variable.

```
100     I% = 1
110     I% = I% + 1
120     PRINT I%
130     IF I% = 100 THEN GOTO 150
140     GOTO 110
150     END
```

HEX$ Function

Purpose:

A string is returned which represents the hexadecimal value of a specified decimal argument.

Version:

LC2, LC4

Format:

v$ = HEX$(x)

x is an integer in the range of -32,767 to 32,767.

Comments:

The HEX$ function is used to convert an integer value to a string representing the hexadecimal equivalent of the value. If x is negative, the two's complement is returned.

Example:

```
100     A% = 16
120     B% = 2047
130     PRINT A%;" DECIMAL OR ";HEX$(A%);" HEX"
140     PRINT B%;" DECIMAL OR ";HEX$(B%);" HEX"
150     END
RUN
16 DECIMAL OR 10 HEX
2047 DECIMAL OR 7FF HEX

ok
```

HIDE Command

Purpose:

Prevents a program from being listed.

Version:

LC4

Format:

HIDE

WARNING!: The only way to restore the LC4 to the normal state and allow programs to be listed is by issuing a NEW command. The NEW command erases the program currently in memory.

IF Statement

Purpose:

Allows conditional execution of a statement based on the result of an expression.

Version:

LC2, LC4

Format:

IF expression THEN statement [ELSE statement]

expression is any numeric expression which will normally be a relational expression.

statement is a BASIC statement or a line number of an existing program line.

Comments:

If the expression is not zero, the condition is considered true and the clause is executed and the ELSE clause is skipped.

If the expression evaluates to zero, the condition is considered false and the THEN clause is skipped and the ELSE clause, if any, is executed. If no ELSE clause is given, execution continues with the next executable statement.

IF... THEN... ELSE statements may be nested. Nesting is limited only by the length of the line. For example:

```
IF X > Y THEN PRINT "GREATER" ELSE IF Y>X THEN PRINT "LESS" ELSE PRINT "EQUAL"
```

is valid.

Example:

The following example illustrates how the IF... THEN... ELSE statement can be used for error checking after calling the OPTOWARE driver:

```
100     GOSUB 1000                      'Call OPTOWARE Driver
110     IF ERRCOD% <> 0 THEN GOSUB 2000
.
.
900     END
1000    CALL OPTOWARE(ERRCOD%,ADDR%,CMD%,POSITIONS%(0),MODIFIERS%(0),INFO%(0))
1010    RETURN
2000    IF ERRCOD% = -1 THEN PRINT "Power Up Clear Expected"
2010    IF ERRCOD% = -2 THEN PRINT "Undefined Command"
2020    IF ERRCOD% = -3 THEN PRINT "Checksum Error"
.
.
2100    IF ERRCOD% = -20 THEN PRINT "Invalid Command"
2110    IF ERRCOD% = -21 THEN PRINT "Invalid Module Position"
2120    IF ERRCOD% = -22 THEN PRINT "Data Range Error"
.
.
2220    IF ERRCOD% = -34 THEN PRINT "Incorrect Echo In 4-Pass"
2230    RETURN
```

INKEY$ Variable

Purpose:

Reads a character from the host communications port (keyboard).

Version:

LC2, LC4

Format:

v$ = INKEY$

Comments:

INKEY$ will only read a single character, even if there are several characters waiting in the communications buffer. The returned value is a null string if there are no characters, or a one character string if a character exists.

While INKEY$ is being used, no characters are echoed back to the screen and all characters are passed through to the program except for a Ctrl-C which stops the program. Pressing the carriage return key in response to INKEY$ will pass an ASCII 13 character to the program.

Example:

The following example illustrates the use of INKEY$ to cause a program to pause until the letter C (upper case c) key is pressed.

```
100     PRINT "Press C to continue"
110     R$ = INKEY$
120     IF R$ <> "C" GOTO 110
```

INP Function

Purpose:

To read a byte from I/O port x.

Version:

LC2, LC4

Format:

v = INP(x)

x is an expression which evaluates to an I/O port number in the range of 0 to 255.

Comments:

Refer to the LC2 or LC4 Hardware Description And Installation manual (Form 217 or 157) for a list of valid I/O port numbers. INP is a complementary function to the OUT statement.

Example:

```
100     DART3 = INP(2)
```

This instruction reads a byte from port 2 and assigns it to the variable DART3.

INPUT Statement

Purpose:

To input data from the host port or the OPTOMUX communications port.

Version:

LC2, LC4

Format:

INPUT[;][#filenum,]["prompt",]variable [,variable,...]

filenum is the logical file number assigned to the communications port or file in a previous OPEN statement. If filenum is omitted, the characters in the host communications buffer are read.

prompt is a string constant or expression which will be sent out the host port to prompt for the desired input. A prompt may be specified only if a filenum is not given.

variable is the name of a string or numeric variable or of an array element for which values are to be input.

Comments:

Upon executing the INPUT statement, a question mark is displayed to indicate that it is waiting for an input. If a "prompt" is used, the prompt string is displayed before the question mark. If a comma is used instead of a semicolon after the prompt string, the question mark is suppressed.

The data entered in response to the INPUT statement must be of the same type as the specified variable. If several items are to be entered, they must be separated by commas and the number of items entered must match the number of variables in the list.

A response to the INPUT statement with too many items, too few items or the wrong type of value causes LC2/LC4 BASIC to issue the message "?Redo from start". If INPUT is immediately followed by a semicolon, the carriage return is suppressed after the input data is entered.

INPUT Statement (Cont.)

Example:

```
100     INPUT R
110     PRINT "The number is: "; R
120     END
RUN
?
```

(The computer is waiting for a response)

```
.
.
?100
The number is: 100

ok
```

In this next example, a prompt is used and the question mark suppressed.

```
100     INPUT "Please enter a number: ", A
110     PRINT "The number is: "; A
120     END
RUN
Please enter a number: 102
The number is: 102

ok
```

INPUT$ Function

Purpose:

Returns a string of x characters, read from the host port, a currently open device, or file buffer.

Version:

LC2, LC4

Format:

v$ = INPUT$(x[,[#]filenum])

x is the number of characters to be read from the buffer.

filenum is the logical file number assigned to the device or file in a previous OPEN statement. If filenum is omitted, the characters in the host communications buffer are read.

Comments:

The INPUT$ function allows all characters to be passed to LC2/LC4 except for a Ctrl-C which is used to interrupt the execution of INPUT$ function. A carriage return does not need to follow the characters sent in response to the INPUT$ function. If a carriage return character is read by the INPUT$ function, it will be treated just like any other character.

Example:

The following example illustrates the use of the INPUT$ function for reading a single character from the host computer or terminal.

```
100     PRINT "ENTER YOUR RESPONSE (Y/N): "
110     X$ = INPUT$(1)
120     IF X$ = "Y" THEN 500
130     IF X$ = "N" THEN 700 ELSE 100
```

The next example uses the INPUT$ function to accept a response from a device such as the Maple Systems, MAP-522 terminal connected to the OPTOMUX network. The example assumes that the OPTOMUX port was previously opened using the OPEN COM... statement.

```
100     X$ = INPUT$(LOC(1),#1)
110     PRINT "MAP 522 RESPONSE: ";X$
```

INSTR Function

Purpose:

This function searches for the occurence of a substring within a string. The position were the substring is found is returned.

Version:

LC2, LC4

Format:

index = INSTR([n], x$, y$)

n - is an optional starting index into the string x$. n is a numeric expression in the range 1 to 255.

x$ - is the string variable to be searched.

y$ - is the substring to search for.

If n is greater than the length of x$ or the substring y$ cannot be found, INSTR will return a 0. If the substring y$ is a null string, the value n will be returned (a 1 will be returned if n was not specified).

Example:

This example finds the end of a line when multiple lines are received using the INPUT$ function:

```
100     A$ = INPUT$(LOC(0), 0)                  'Get All From Host Port
110     INDEX% = INSTR( A$, CHR$(13))           'Look For Carriage Return
120     IF INDEX% = 0 THEN GOTO 1000
```

INT Function

Purpose:

Returns the nearest integer that is smaller than or equal to x.

Version:

LC2, LC4

Format:

v% = INT(x)

x is any numeric expression, usually real.

Comments:

The main differences between CINT, INT, and FIX are as follows: CINT rounds a number to the nearest integer, INT returns the largest integer less than or equal to the number, and FIX removes the fractional part of a number without rounding.

Example:

```
PRINT INT(53.89)
53

ok
PRINT INT(-1.66)
-2

ok
```

Notice that INT truncates positive numbers but rounds negative numbers towards a greater negative value.

KILL Command

Purpose:

Deletes a file from the RAM disk area of the LC4.

Version:

LC4

Format:

KILL filespec

filespec can be any 12 printable characters enclosed in quotes or the string "*.*". If "*.*" is used, all files on the RAM disk will be deleted. No wildcard characters are allowed in the filespec.

Example:

The following line deletes a file named TEMPS.ZONE1 in the RAM disk area of the LC4.

```
KILL "TEMPS.ZONE1"
```

LEFT$ Function

Purpose:

Returns a substring which consists of the left-most n characters of string x$.

Version:

LC2, LC4

Format:

v$ = LEFT(x$, n)

x$ is any string expression.

n is a numeric expression which specifies the number of left-most characters to be returned. n must be in the range 0 to 255.

Comments:

If n is greater than the length of the string x$, the entire string x$ is returned. If n = 0, the null string (length zero) is returned.

Example:

In this example, the LEFT$ function is used to extract the first 7 characters in the string INFO$

```
100     INFO$ = "OPTO 22, HUNTINGTON BEACH, CALIFORNIA"
110     NAME$ = LEFT$(INFO$,7)
120     PRINT NAME$
RUN
OPTO 22

ok
```

LEN Function

Purpose:

Returns the number of characters in the string x$.

Version:

LC2, LC4

Format:

v% = LEN(x$)

x$ is any string expression.

Comments:

The number of characters returned includes non-printable characters and blanks.

Example:

```
100     X$ = "NEWPORT BEACH, CA"
110     PRINT LEN(X$)
RUN
17

ok
```

There are 17 characters in the string "NEWPORT BEACH, CA", because the comma and the blank are counted.

LET Statement

Purpose:

The value of an expression is assigned to a specified variable.

Version:

LC2, LC4

Format:

[LET] variable = expression

variable is a string or numeric variable or array element which is to receive the value determined by expression.

expression is an expression which LC2/LC4 BASIC evaluates and assigns to variable. The type of the expression (string or numeric) must match the type of the variable, or a "type mismatch" error will occur.

Comments:

The use of the word LET is optional. The equal sign is sufficient when assigning a value to variable.

Example:

This example assigns the value 100 to the variable XYZ. It then assigns the value 110, which is the value of the expression XYZ + 10, to the array variable F(1). The string "BOARD 1" is assigned to the variable MUX$.

```
100     LET XYZ = 100
110     LET F(1) = XYZ + 10
120     LET MUX$ = "BOARD 1"
```

The same statements could also have been written:

```
100     XYZ = 100
110     F(1) = XYZ + 10
120     MUX$ = "BOARD 1"
```

LINE INPUT Statement

Purpose:

Reads an entire line (up to 255 characters) from the host port, ignoring commas or other delimiters.

Version:

LC2, LC4

Format:

LINE INPUT [;]["prompt";] stringvar

prompt is a string constant that is sent to the host before any input is accepted. In order for a question mark to be printed, it must be part of the prompt string.

stringvar is the name of the variable (string or string array element) to which the incoming line will be assigned. All input from the end of the prompt to a carriage return will be assigned to stringvar.

Comments:

The LINE INPUT statement works identically to the INPUT statement but also allows delimiters and control charactes to be passed to the variable. The host can exit the LINE INPUT statement by passing a Ctrl-C. Upon receiving the Ctrl-C, LC2/LC4 BASIC will return to the command level and display "ok". A CONT can then be issued to resume execution at the LINE INPUT statement.

Example:

This example prompts the host to send a list of messages to be stored in an array, then waits for each message line to be sent by the host. Each line is assigned to one element of the array ERRMSG$ and each line must be followed by a carriage return.

```
100     DIM ERRMSG$(15)
110     LINE INPUT "Send list of 16 error messages: "; ERRMSG$(0)
120     FOR I% = 1 TO 15
130     LINE INPUT ERRMSG$(I%)
140     NEXT
```

LIST, LLIST Command

Purpose:

Lists the program currently in memory to the host port or to the OPTOMUX communications port when LLIST is used.

Version:

LC2, LC4

Format:

LIST [line1] [-[line2]]

LLIST [line1] [-[line2]]

line1, line2 are valid line numbers in the range 0 to 32,767. line1 is the first line to be listed. line2 is the last line to be listed.

Comments:

If the line range is omitted, the entire program is listed.

When the dash (-) is used in a line range, three options are available:

If only line1 is given, then all the lines starting at line1 until the end of the program are listed.

If only line2 is given, then all the lines starting from the first line in the program, until and including line2, are listed.

If both line numbers are specified, all lines from line1 through line2, inclusive, are listed.

LLIST routes the program listing to the OPTOMUX communications port instead of the host port. This is useful for making printed listings when a printer is attached to the OPTOMUX serial port.

Example:

LIST	Lists the entire program on the screen.
LIST 1000-	Lists all lines from 1000 through the end of the program.
LIST -2000	Lists all lines from starting at 0 to 2000, inclusive.
LIST 100-1520	Lists all lines from 100 to 1520, inclusive.

LOC Function

Purpose:

Returns the number of charcters in the OPTOMUX input buffer.

Version:

LC2, LC4

Format:

v = LOC(filenum)

filenum is the logical file number used when the OPTOMUX communications port was opened.

Comments:

The LOC function returns the number of characters in the OPTOMUX input buffer that are waiting to be read. The input buffer can hold up to 255 characters.

Example:

The following example waits until there are 112 characters in the OPTOMUX input buffer, then assigns the contents of the buffer to the variable MESSG$. The example assumes the OPTOMUX port has been previously opened as logical file 1.

```
100     IF LOC(1) >= 112 THEN GOTO 120
110     GOTO 100
120     MESSG$ = INPUT$(112,#1)
```

LOF Function

Purpose:

Returns the number of free space (bytes) in the communications input buffer.

Version:

LC2, LC4

Format:

v = LOF(filenum)

filenum is the logical file number used when the communications port was opened.

Comments:

The LOF function returns the amount of free space calculated to be 255 minus LOC(filenum). This function is useful for checking when the OPTOMUX input buffer is almost full.

Example:

The following example uses the LOF function to indicate when the input buffer is full so the contents of the buffer can be assigned to several variables using the INPUT$ function. The example assumes that the OPTOMUX port was previously opened as logical file #1.

```
100     IF LOF(1) > 0 THEN GOTO 100
110     FOR I% = 1 TO 4
120     MESSG$(I%) = INPUT$(64,#1)
130     NEXT
```

LOG Function

Purpose:

Calculates the natural log of a numeric expression.

Version:

LC2, LC4

Format:

v = LOG(x)

x is any numeric expression greater than zero.

Example:

Print the natural logs of all the integers between 1 and 100.

```
100     FOR I% = 1 TO 100
110     PRINT "Log of ";I%;"is "; LOG(I%)
120     NEXT
```

MID$ Function

Purpose:

Returns the requested part of a specified string.

Version:

LC2, LC4

Format:

v$ = MID$ (x$, m, [n])

x$ is any string.

m is an integer expression in the range of 1 to 255.

n is an integer expression in the range of 0 to 255.

This function returns a string of length n beginning with the m character. If n is omitted or is less than the remaining number of characters, the remaining right-most characters are returned. If n is zero or n is greater than LEN(x$), then MID$ will return a null string.

Example:

This example grabs the OPTOMUX address from a command string.

```
100     COMM$ = MID$(OCMD$, 2, 2)              'Get Address From String
110     IF COMM$ <> "FF" THEN GOTO 1000        'If Not Address 255
```

NEW Command

Purpose:

The program currently in memory is deleted and all variables are cleared.

Version:

LC2, LC4

Format:

NEW

Comments:

The NEW command must be issued prior to entering a new program or after power up with no existing program in memory. LC2/LC4 BASIC will always return to the command level after the NEW command is executed.

Example:

The program memory will be deleted and all the program variables are cleared.

```
NEW

ok
```

ON COM(n) GOSUB Statement

Purpose:

Branches to a subroutine when a character is received on the specified serial port.

Version:

LC2, LC4

Format:

ON COM(n) GOSUB line

n is a numeric expression which specifies the serial port to trap on.

line is the beginning line number of the subroutine to branch to when a character is received on port n.

Comments:

In order to activate the ON COM statement, a COM(n) ON statement must be used to activate and trap the communications interrupt. After COM(n) ON and if a non-zero line number is specified in the ON COM(n) GOSUB statement, then at the end of executing a program line, BASIC will check to see if any characters have been received in the specified port. If so, BASIC will perform a GOSUB to the specified line.

Issuing a COM(n) OFF statement will disable the ON COM(n) GOSUB feature and further characters will not cause an ON COM interrupt.

If a COM(n) STOP statement is used after a COM(n) ON statement, the ON COM(n) GOSUB feature will be disabled; but if a character is received, the interrupt will be held pending until a COM(n) ON statement is executed. Once the COM(n) ON statement is executed, the subroutine of the ON COM(n) GOSUB statement will be executed if the interrupt was pending.

When the COM feature interrupts and branches to the subroutine, an automatic COM(n) STOP is executed.

Acceptable COM port numbers are 0 through 3. COM0 is the host port (which is opened by default). COM ports 2 and 3 are on the EX2 daughter card and can only be accessed when using a LC4 with an EX2 daughter card installed. The LC2 has only COM port number 0 and 1 available.

Example:

```
100   ON COM(1) GOSUB 2000          'Specify Interrupt line
110   COM(1) ON                     'Enable ON COM Interrupt
.
.
1000  A$ = INPUT$(LOC(1),1)         'Read Received Character
1010  RETURN                        'Return To Calling Subroutine
```

ON ERROR GOTO Statement

Purpose:

Branches to a subroutine when a BASIC error occurs.

Version:

LC2, LC4

Format:

ON ERROR GOTO line

line is the line number of the beginning of the subroutine to branch to when a BASIC error has been detected.

Comments:

While BASIC is in the error trapping routine, error trapping is disabled. Error trapping will be restarted after a RESUME statement is executed.

ON ERROR trapping can be disabled by executing an ON ERROR GOTO 0 statement.

Example:

This example tests to see if an overflow error has occurred.

```
100     ON ERROR GOTO 1000
.
.
.
1000    IF ERR <> 6 THEN GOTO 1030          'Go Around If Not Over
1010    ANSWER% = 32767                     'Set Highest Integer
1020    RESUME NEXT                         'And Do Next Line
```

ON... GOSUB And ON... GOTO Statements

Purpose:

Allows branching to one of several specified line numbers depending on the value of an expression.

Version:

LC2, LC4

Format:

ON n GOSUB line [,line,...]

ON n GOTO line [,line,...]

n is a numeric expression in the range of 0 to 255 which is rounded to an integer value if necessary.

line is the line number of the line to branch to.

Comments:

The value of n is used in determining which line number in the list the program will branch to. If the value of n were to evaluate to 4, the program would branch to the fourth line number in the list. If n evaluates to 0 or larger than the number of items in the list, then LC2/LC4 BASIC will continue executing at the program line which follows the ON... GOTO or ON... GOSUB statement. If n is outside the specified range, an "illegal function call" error will occur.

The ON... GOSUB statement will branch to a subroutine that must have a RETURN statement to return execution to the line following the ON... GOSUB statement.

Example:

The first example will branch to line number 200 if the expression X*3/5 evaluates to 1, branch to line 300 if expression equals 2, line 400 if expression equals 3, and line 500 if expression equals 4. If X*3/5 is equal to zero or greater than 4, but still within the range of 1 to 255, then the program will continue executing with the next statement.

```
100     ON X*3/5 GOTO 200, 300, 400, 500
```

The next example shows the use of the ON... GOSUB statement.

```
1000    ON P GOSUB 3000, 4000, 5000
.
.
3000    '*   Start Of Subroutine For P=1
.
.
3999    RETURN
.
.
.
```

ON KEY(n) Statement

Purpose:

Branches to a subroutine at a specified line when a trapped character is received at the host communications port.

Version:

LC2, LC4

Format:

ON KEY(x$) GOSUB line

x$ is a string expression of which the first character is used to compare with any characters received at the host communications port.

line is the line number of the beginning of the subroutine to branch to when an incoming character matches the first character in x$.

Comments:

The ON KEY function can trap up to 16 characters or "levels" using individual ON KEY(x$) GOSUB statements to specify each character and its associated subroutine. All standard, 7 bit ASCII characters can be used for trapping.

In order to activate the ON KEY function, a KEY(x$) ON statement must first be used. A KEY(x$) ON statement is needed for each character that is to be trapped. After the KEY(x$) ON statement is executed and a line number is specified in the ON KEY... GOSUB statement, LC2/LC4 BASIC will branch to that line number whenever a character which is equal to the first character in x$ appears at the host communications port. LC2/LC4 BASIC will branch to the subroutine only after the current line is executed.

If a KEY(x$) OFF statement is executed, no trapping takes place for the specified character. Even if the character appears at the host communications port, the event is not remembered.

If a KEY(x$) STOP statement is executed, no trapping takes place for the specified character. However, if the first character specified in x$ appears at the host communications port, the event is remembered, so an immediate trap will occur when KEY(x$) ON is executed.

When a branch is made to the subroutine after a character is matched, an automatic KEY(x$) STOP is executed so recursive traps can never take place. The RETURN from the subroutine automatically does a KEY(x$) ON unless an explicit KEY(x$) OFF was performed inside the subroutine.

ON KEY(n) Statement (Continued)

Example:

The following is an example of a trap routine for the characters "C", "A", and ">". Notice how variables with values having more than one character can be used. Only the first character is used for trapping.

```
100     ON KEY("C") GOSUB 1000
110     KEY("C") ON
120     CHAR$ = ">Hello"
130     ON KEY(CHAR$) GOSUB 2000
140     KEY(CHAR$) ON
150     B$ = "A"
160     ON KEY(B$) GOSUB 3000
170     KEY("A") ON
.
.
1000    '*    Routine For "C" At Host Port
.
.
1599    RETURN
.
.
2000    '*    Routine For ">" At Host Port

.
2599    RETURN
.
.
3000    '*    Routine For "A" At Host Port
.
.
3599    RETURN
```

ON TIMER Statement

Purpose:

Branches to a subroutine when a specified period of time has elapsed.

Version:

LC2, LC4

Format:

ON TIMER (n) GOSUB line

n is a numeric expression in the range of 0.1 to 3276.7 for the LC2 and 0.01 to 42,949,669.99 for the LC4 which represents time in seconds. A fractional value will specify fractions of a second down to a resolution of 100 milliseconds for the LC2 and 10 milliseconds for the LC4. A value entered that is outside this range results in an "illegal function call" error.

line is the beginning line number of the subroutine to branch to whenever n seconds has elapsed.

Comments:

In order to activate the ON TIMER function, a TIMER ON statement must be used. After a TIMER ON statement is executed and a line number is specified in the ON TIMER... GOSUB statement, LC2/LC4 BASIC will branch to that line number after the specified time has elapsed.

Although the minimum time that can be specified is 100 ms (n = 0.1) for the LC2 or 10 ms (n=0.01) for the LC4, a large ON TIMER subroutine may take longer than 100 (or 10) milliseconds to execute. The result is that after the routine is finished, a previously pending ON TIMER interrupt will cause the routine to execute again. LC2/LC4 will then be in a loop which constantly executes the ON TIMER subroutine and nothing else. Therefore, one must use care when deciding the amount of time to specify for servicing the ON TIMER subroutine.

Issuing a TIMER OFF statement will disable the ON TIMER GOSUB feature and further timing interrupts are not remembered.

If a TIMER STOP statement is used after a TIMER ON statement, the ON TIMER GOSUB feature will be disabled but timing activity continues and interrupts are held pending until a TIMER ON statement is executed. Once the TIMER ON statement is executed, the subroutine of the ON TIMER GOSUB statement will be executed if the interrupt was pending. The TIMER ON statement resets the ON TIMER down counter to its initial valvue.

When the timer feature interrupts and branches to the subroutine, an automatic TIMER STOP is executed so that recursive interrupt handling never takes place. The RETURN from the interrupt service routine does an effective TIMER CONTINUE (the timer is not reset to initial value), unless and explicit TIMER OFF was performed inside the subroutine.

ON TIMER Statement (Continued)

Example:

The following example illustrates how an OPTOMUX station can be polled every 400 milliseconds to retrieve alarm status conditions. The OPTOMUX station is assumed to be a 4 point, discrete I/O board at address 255. If an alarm is active, FAULTDEV will contain a value other than 0.

```
100     ON TIMER (0.4) GOSUB 500
110     TIMER ON
.
.
500     '*   Timer Interrupt Service Routine
510     ADDR% = 255
520     CMD% = 64
530     CALL OPTOWARE (ERRCOD%, ADDR%, CMD%, POS%(0), MOD%(0), INFO%(0))
540     FAULTDEV = INFO%(0) AND &H0F
550     RETURN
```

OPEN Statement

Purpose:

Selects or creates a file in the RAM disk area of memory for access.

Version:

LC4

Format:

OPEN filespec FOR mode AS [#]filenum

filespec can be any 12 printable characters enclosed in quotes.

mode is one of the following:

APPEND	specifies sequential output mode with the file pointer positioned at the end of the file.
INPUT	specifies sequential input mode.
OUTPUT	specifies sequential output mode.

mode is a string constant not enclosed in quotation marks.

filenum is an integer expression from 1 to 7.

Comments:

The OPEN statement only supports sequential access. The same file may be opened under different filenums for INPUT, but can only be opened once for OUTPUT or APPEND. A file may be opened for INPUT and OUTPUT (or APPEND) using different filenums only.

Opening a file for OUTPUT does not erase the file, but positions the file pointer at the start of the file. A subsequent write to the file will overwrite the original data. To completely erase a file see the KILL command. If a file opened for OUTPUT or APPEND does not exist, a new file will be created. If a file opened for INPUT does not exist, a "file not found" error will occur.

To write to a file, use the PRINT # command; to read data from a file, use the INPUT$ or the INPUT # statements.

The LC4 COM port 0 has a default filenum of 0. This filenum is not accessible with the OPEN statement but indicates that COM0 is already open by default. The COM port 0 can never be closed.

Example:

In this example, a file called CONFIG.FILE is created and the values of the array CONFIG% are stored.

```
100     OPEN "CONFIG.FILE" FOR OUTPUT AS #1
110     FOR I% = 0 TO 9
120     PRINT #1,CONFIG%(I%)
130     NEXT
140     CLOSE #1
```

OPEN "COM... " AS #f Statement

Purpose:

Opens a communications port and assigns it to a logical file number.

Version:

LC2, LC4

Format:

OPEN "COMn:[speed] [,parity] [,data] [,stop]" AS [#] filenum

n is the number of the communications port to open. When using a LC2/LC4, the host port is considered port number 0 and the OPTOMUX port is port number 1. The LC4 also supports port numbers 2 and 3 when an EX2 daughter card is installed.

speed this parameter is used by IBM/Microsoft BASIC to set the baud rate of the communications port. However, since the baud rate setting for LC2/LC4 is defined by hardware jumpers, this parameter is ignored by LC2/LC4 BASIC.

parity is a one character constant which can be one of the following characters:

E: EVEN	Parity bit is always even.
N: NONE	No parity is used.
O: ODD	Parity bit is always odd.

data this is a number indicating the number of transmit and receive data bits. Acceptable values are 5, 6, 7, or 8. The LC4 only allows 7 and 8 data bits on COM0 and COM1.

stop this number indicates the number of stop bits in the character. Acceptable values are a 1 or 2. Selecting 5 data bits and 2 stop bits will result in an actual stop bit value of 1 1/2.

filenum is an integer expression which evaluates to a logical file number. This number is then associated with the specified communications port for as long as it is open. Multiple filenums can be associated with the same serial port at one time. A user filenum can be any value from 1 to 7. Note that on power up, the host port is automatically opened with a filenum of 0 (this filenum cannot be closed by the user). The default values for the OPTOMUX port are: No parity, 8 data bits, and 1 stop bit.

Comments:

The OPEN "COM... " statement initializes the communications port when other than standard type devices are attached to this port. On power up, all serial communications ports are set for no parity, 8 data bits, and 1 stop bit. IBM/Microsoft BASIC's line signal options CS, DS, CD, and LF are not used by LC2/LC4 and, if present, are ignored. When specifying the RS option, the OPTOMUX port will only be enabled when being written to. The RS option is necessary when multiple hosts are connected to LC2/LC4. If RS is not specified, the port will always be enabled. The communications ports may be assigned to more than one file number at a time.

Example:

The OPTOMUX communications port is assigned to logical file number one (1). The baud rate is determined by the hardware jumpers on the LC2/LC4 board and there will be no parity, eight data bits, and one stop bit.

```
100     OPEN "COM1:19200,N,8,1" AS #1
```

OUT Statement

Purpose:

Sends a byte to a LC2/LC4 output location.

Version:

LC2, LC4

Format:

OUT m, n
m is a numeric expression for the port number in the range of 0 to 255.
n is a numeric expression for the data to be transmitted in the range of 0 to 255.

Comments:

OUT is the complementary statement to the INP function. Refer to the LC2 and LC4 Hardware Description and Installation manuals (Forms 217 or 157) for descriptions of valid I/O addresses.

Example:

This sends the value 0 to output port 16.

```
100     OUT 16,0
```

PEEK Function

Purpose:

Returns a byte which is stored at the specified memory location.

Version:

LC2, LC4

Format:

v = PEEK(n)
n this is an integer in the range -32,768 to 32,767 (0 to FFFF Hex) which indicates the address of the memory location to be read.

Comments:

The returned value will be an integer in the range 0 to 255.

PEEK is the complementary function to the POKE statement.

Example:

The following example calculates an address by summing the variables VAL1%, VAL2%, and VAL3% and putting the contents of that location in the variable SUMBAL.

```
100    VAL1% = 3021
110    VAL2% = 104
120    VAL3% = 1000
130    SUMVAL = PEEK(VAL1% + VAL2% + VAL3%)
```

POKE

Purpose:

Writes the specified byte into the specified memory location.

Version:

LC2, LC4

Format:

POKE m,n

m is the address of the memory location where the data is to be written and must be in the range of -32,768 to 32,767 (0 to FFFF Hex).

n is the byte of data to be written to location m and must be in the range of 0 to 255.

Comments:

POKE and PEEK are complementary functions and are useful for loading machine language routines, passing parameters to and from these machine language subroutines or for data storage.

WARNING!: BASIC does not do any checking on the address, so POKEing around in BASIC's stack, BASIC's variable area, or your BASIC program may cause your LC2/LC4 to behave erratically.

POKEing into the memory locations where the ROMs reside will not cause any changes in memory. These devices are "read only" and cannot be written to. The ROM space exists from 0 to 7FFF Hex 0 the LC2 and LC4. The RAM space starts from 8000 to FFFF Hex.

Example:

```
100   VAL1% = &H8001
110   VAL2% = 38
120   DATAVAL = PEEK (VAL1%)
130   ADDRVAL = VAL1% + VAL2%
140   POKE ADDRVAL, DATAVAL
```

PRINT, LPRINT Statement

Purpose:

To write data to either the host communications port or a logical device or file.

Version:

LC2, LC4

Format:

PRINT[[#]filenum,][list of expressions][;]

LPRINT [list of expressions][;]

list of expressions is a list of one or more valid numeric or string expressions separated and/or terminated by a comma or semicolon delimiter.

filenum is the logical file number assigned to the file or logical device opened with a previous OPEN statement. If filenum is omitted, the characters printed are sent to the host port.

Comments:

A semicolon (;) causes the next value to be printed immediately after the last value. A semicolon at the end of an expression list suppresses the carriage return.

A comma (,) causes an ASCII TAB character to be output. When printed to the host or to a printer, the TAB character will move the print "cursor" to the next tab column (normally located every 8 character positions). A comma at the end of a list suppresses the carriage return and moves the print "cursor" to the next tab column.

NOTE: LC2/LC4 BASIC requires the use of a semicolon or comma to separate variables. Use of spaces as delimiters will generate an error message. This is a deviation from IBM/Microsoft BASIC.

LPRINT is identical to PRINT except the output is routed to the OPTOMUX communications port instead of the host port.

Example:

```
100     PRINT "HELLO";
110     PRINT " WORLD"
120     PRINT 2+2, 3+4, 5*6, "HA HA HA"
RUN
HELLO WORLD
4       7    30         HA HA HA
```

READ Statement

Purpose:

Retrieves information from the data list set up by the DATA statement (also see the DATA statement).

Version:

LC2, LC4

Format:

READ variable [,variable,...]

variable is a numeric or string variable or array element which will receive the value read from the DATA list.

Comments:

Values in a DATA statement are assigned to the variables in the READ statement on a one-to-one correspondence. The variables in the READ statement can be numeric or string types but must match the type of values in the DATA statement being read, otherwise, an error will occur.

One or more DATA statements may be accessed in order by a single READ statement. If there are more variables in the READ statement than there are elements in the DATA statement, an "out of data" error will occur. Having fewer data elements in the DATA statement than there are variables in the READ statement will cause any subsequent READ statements to begin reading data at the first unread element. Extra elements in the DATA statement are ignored if there are no more READ statements in the program.

The RESTORE statement allows the data pointer to be placed at the beginning of the first DATA statement in a program or at a DATA statement at a specified line (also see the RESTORE statement).

Example:

This program reads string and numeric data from the DATA statement in line 120. Notice that quotation marks are not needed around Samuel. Quotations marks are needed around "Jones," because of the comma.

```
100     PRINT "LAST", "FIRST", "PHONE"
110     READ L$, F$, P
120     DATA "Jones,",Samuel,5551934
130     PRINT L$,F$,P
RUN
LAST    FIRST         PHONE
Jones,  Samuel        5551934

ok
```

REM Statement

Purpose:

Inserts comments in a program.

Version:

LC2, LC4

Format

REM remark

remark may be any sequence of characters.

Comments:

A REM statement tells LC2/LC4 BASIC to disregard any characters after the REM and up to the end of the line. The characters are stored in memory for listing purposes, but are non-executable.

REM statements may be placed anywhere in a program and may be branched to using the GOTO or GOSUB statements. In such a case, execution will continue with the first executable statement after the REM statement.

A single quotation mark may be used in place of the REM statement. If a REM statement is placed in a line containing other program statements, the REM statement must be the last statement on the line.

Example:

```
100     REM          Calculate Speed
110     HOURS = 1                                   REM      Initialize HOURS
120     INPUT "Distance(miles) is: ";DISTANCE       'Prompt For Distance
130     INPUT "Time (hours) is: ";HOURS             'Prompt ForTime
140     REMNow Calculate And Print The Speed
150     PRINT "Speed (mph) is: "; DISTANCE/HOURS
```

RESTORE Statement

Purpose:

Allows DATA statements to be reread from a specified line.

Version:

LC2, LC4

Format

RESTORE [line]

line is the line number of a DATA statement in the program.

Comments:

The RESTORE statement will reset the DATA pointer to the first DATA statement in the program so it can be accessed by a subsequent READ statement. If line is specified, the next READ statement accesses the first item in the DATA statement at the specified line. If a DATA statement does not exist at line, the DATA pointer is set to the first DATA statement encountered after line.

Example:

```
100     READ X,Y,Z
110     RESTORE
120     READ T,U,V
130     DATA 42, 78, 31
140     PRINT X; Y; Z; T; U; V
RUN
42 78 31 42 78 31

ok
```

The RESTORE statement in line 110 resets the DATA pointer to the beginning, so that the values that are read in line 130 are 42, 78, and 31.

RESUME Statement

Purpose:

Continues program execution after an error recovery routine is performed.

Version:

LC2, LC4

Format:

RESUME [line]

Comments:

line can be any of three values: 0, the actual line to perform, or the word NEXT.

RESUME or RESUME 0 will resume execution at the line that caused the error.

RESUME line will resume execution at the specified line number.

RESUME NEXT will resume execution at the line following the line that caused the error.

A RESUME statement executed that is not in an error recovery routine will cause a "RESUME without error" message to occur.

Example:

The following example is used to set a high value when an overflow condition occurs:

```
100     ON ERROR GOTO 1000
.
.
.
1000    IF ERR <> 6 THEN GOTO 4020          'Go Around If Not Overflow
1010    ANSWER% = 32767                     'Set High Limit
1020    RESUME NEXT                         'Skip Error Line
```

RETURN Statement

Purpose:

To exit a subroutine called with the GOSUB statement. Also see the GOSUB statement.

Version:

LC2, LC4

Format:

RETURN

Comments:

The RETURN statement will exit a subroutine, and begin execution at the line immediately following the most recently executed GOSUB statement. A subroutine may have several RETURN points. Execution of a RETURN without a prior GOSUB will result in a "RETURN without GOSUB error".

Example:

```
100     GOSUB 1000
.
.
900     END
.
.
.
1000    IF C$ = "ERROR" THEN RETURN
1010    IF C$ = "LOW" THEN A = 10
1020    IF C$ = "HIGH" THEN A = 100 ELSE A = 45
1030    B = A * 5.145 / D
1040    RETURN
```

RIGHT$ Function

Purpose:

Returns a subset of x$ which consists of the right-most n characters of that string.

Version:

LC2, LC4

Format:

v$ = RIGHT(x$,n)
x$ is any string expression.
n is an integer expression which specifies the number of characters to be returned.

Comments:

If n is greater than or equal to the length of x$, then the entire string x$ is returned. If n is zero, the null string (length zero) is returned.

Example:

The right-most ten characters of the string B$ are returned.

```
100     B$ = "OPTO22, Huntington Beach, California"
110     PRINT RIGHT$(B$,10)
RUN
California

ok
```

RUN Command

Purpose:

Begins execution of a program.

Version:

LC2, LC4

Format:

RUN

Comments:

RUN begins execution of the program currently in memory starting at the lowest line number.

Example:

The following example illustrates the use of the RUN command to execute a short program.

```
100     REM This program prints the numbers from 1 to 10
110     FOR A% = 1 TO 10
120     PRINT A%;
130     NEXT
RUN
1 2 3 4 5 6 7 8 9 10

ok
```

SIN Function

Purpose:

Returns the sine of the argument value which is expressed in radians.

Version:

LC2, LC4

Format:

v = SIN(x)

x is the angle in radians whose sine is to be calculated.

Comments:

The result returned by the sine function is a single precision, real value. To convert a value in degrees to radians, multiply the number of degrees by 0.01745329.

Example:

```
100     DEG2RAD = 0.01745329                'Obtained From 3.14159/180
110     DEGREES = 90
120     RADIANS = DEGREES * DEG2RAD
130     PRINT SIN(RADIANS)
RUN
1

ok
```

This example converts 90 degrees to its equivalent radian value then calculates the sine.

SLEEP Command

Purpose:

To disable a single LC2/LC4 or all LC2/LC4's on a multidropped communications line in order to download programs to selected units.

Version:

LC2, LC4

Format:

SLEEP [n]

n is an integer value which indicates the address of the LC2/LC4 unit to be disabled.

Comments:

If no address is specified when the SLEEP command is issued, all LC2/LC4 units on the same communications link which are powered up and not executing a program will be disabled from the communications line.

The SLEEP command is normally issued from the host terminal or computer when different programs must be downloaded to separate LC2/LC4's. After all LC2/LC4's are disabled, individual units can be made active using the AWAKE command to enable a unit and then download or enter a program from the host terminal.

Also see the AWAKE command and ADDRESS@ variable.

Example:

See the APPENDICES section on Multidropping Several LC2/LC4's.

SQR Function

Purpose:

Returns the square root of x.

Version:

LC2, LC4

Format:

v = SQR(x)

x must be greater than or equal to zero.

Example:

This example calculates the square roots of the numbers 100, 300, 500, 700, and 900.

```
100     FOR A = 100 TO 900 STEP 200
110     PRINT A; SQR(A)
120     NEXT
RUN
100 10
300 17.320508
500 22.360679
700 26.457513
900 30

ok
```

STOP Statement

Purpose:

Terminates program execution and returns to the command level.

Version:

LC2, LC4

Format:

STOP

Comments:

STOP statements may be inserted anywhere in a program to terminate execution. When LC2/LC4 BASIC encounters a STOP statement, the following message will be displayed:

```
break in line nnnnn
```

where nnnnn is the line number where the STOP occurred.

Program execution may be resumed by issuing a CONT statement at the command level.

Example:

This example prints the message "HELLO WORLD" then stops. The CONT statement allows the program to continue and the value of A is then printed.

```
100     PRINT "HELLO WORLD"
110     A = 10 * 1.45
120     STOP
130     PRINT "A = "; A
RUN
HELLO WORLD
break in line 120


ok
CONT
A = 14.5

ok
```

STR$ Function

Purpose:

Converts x to its equivalent string value.

Version:

LC2, LC4

Format:

v$ = STR$(x)

x is any numeric expression.

Comments:

The string returned by STR$ will contain a leading blank if x is positive. This leading space is the space reserved for the plus sign (+). Large numbers and small numbers will be converted to exponential form. Fractional values will generate leading zeros if not converted to exponential form. The VAL function is complementary to the STR$ function.

Example:

This example first converts the number in A to a string B$, then prints the value of the string and its length.

```
100     A = 123456
110     B$ = STR$(A)
120     PRINT B$; LEN(B$)
RUN
123456 7
```

TIME$ Variable And Statement

Purpose:

Sets or returns the current time from LC2/LC4's real-time clock.

Version:

LC2, LC4

Format:

v$ = TIME$
or
TIME$ = x$

Comments:

The current time is returned as a string which is eight characters long. The string is returned as hh:mm:ss, where hh is the hour (00 to 23), mm is the minutes (00 to 59), and ss is the seconds (00 to 59).

For the statement TIME$ = x$, used for setting the current time. x$ is a string expression which indicates the time to be set. x$ may be specified in either of the following forms.

hhSets the hour in the range 0 to 23. Minutes and seconds default to 00.

hh:mmSets the hour and minutes. Minutes must be in the range 0 to 59.
Seconds default to 00.

hh:mm:ssSets the hour, minutes, and seconds. Seconds must be in the range 0 to 59.

At least one digit must be included in the above forms and the leading zero may be omitted. For example, if you wanted to set the time as 43 minutes after midnight, you could enter TIME$ = "0:43", but not TIME$ = ":43". Out of range values will cause an "illegal function call" error and the previous time is retained. If x$ is an illegal string, a "type mismatch" error will occur.

Example:

The following example is a subroutine which will send the host an alarm message and the time of occurrence assuming OPTOMUX has just detected the error.

```
100     ALLMSG$ = "Pump 5 failure at"
110     PRINT ALLMSG$; TIME$
120     RETURN
```

TIMER Function

Purpose:

Returns a floating point number indicating the number of seconds since midnight.

Version:

LC2, LC4

Format:

v = TIMER

Comments:

The resolution of the TIMER function is 100 milliseconds for LC2 and a resolution of 1 second for LC4.

Example:

This example pauses a BASIC program until the current time is 3:55:00 AM.

```
100     IF TIMER < 14100 THEN GOTO 100          'Wait For 03:55:00
```

TROFF And TRON Commands

Purpose:

To trace the execution of program lines.

Version:

LC2, LC4

Format:

TROFF

TRON

Comments:

The TRON command may be used in the indirect mode for purposes of debugging program lines. TRON causes the line number of each program line to be printed as the line is executed. The line numbers will be displayed enclosed in square brackets. The trace feature is turned off by using the TROFF command. The TRON and TROFF statements may also be used within a BASIC program.

Example:

In this example, the trace feature is activated by issuing the TRON statement before the program is run. As the program is executed, the line numbers appear in brackets. The numbers not in brackets are the values of P, M, and Q which are printed by the program.

```
100     M = 100
110     FOR P = 1 TO 3
120     Q = M + 50
130     PRINT P; M; Q
140     M = M + 100
150     NEXT
160     END
TRON

ok
RUN
[100][110][120][130] 1  100  150
[140][150][120][130] 2  200  250
[140][150][120][130] 3  300  350
[140][150][160]

ok
TROFF

ok
```

VAL Function

Purpose:

Converts the string x$ to its numerical value.

Version:

LC2, LC4

Format:

v = VAL(x$)

x$ is a string expression.

Comments:

Leading blanks, line feeds, and tabs are stripped off by the VAL function before the string x$ is converted. The VAL function will stop converting when a non-numeric character is encountered in x$. The complement of the VAL function is the STR$ function.

Example:

In this example, VAL is used to extract the house number from an address.

```
PRINT VAL("15461 Springdale Street")
15461

ok
```

VARPTR Function

Purpose:

Returns the address in memory of the specified variable.

Version:

LC2, LC4

Format:

v = VARPTR(variable)

variable is the name of a numeric or string variable or array element in your program.

Comments:

The address returned by the VARPTR function is an integer in the range 0 to 65,535. When using VARPTR to return the address of an array element, specifying element zero (VARPTR(A(0))) will return the base address of the array. This is useful when passing parameters to machine language programs because the other elements in the array can be calculated from the base address.

WARNING!: The VARPTR function should only be called when it is needed. Assigning the result of a VARPTR function to a variable, and then using that variable later in the program may produce the wrong results. A variable's location may move around as other variables grow or change. The following is a table of the different types of variable and its byte size.

Type	Size
Integer	2 bytes
Real	4 bytes
String	1 byte per character

Integer values are stored least significant byte first. Arrays and strings are stored sequentially.

Example:

This example uses the VARPTR function to get the data from a variable. This is not a very practical example but it does illustrate how the VARPTR function works. Line 110 obtains the address of the variable DATA% by using the VARPTR function. Since DATA% is an integer, its data is stored as two bytes with the least significant byte first. Line 120 gets the actual value of the data by reading the values using the PEEK function. The value in the second position is multiplied by 256 because it is the high order value.

```
100     DATA% = 500
110     ADDR = VARPTR(DATA%)
120     VALUE = PEEK(ADDR) + 256 * PEEK(ADDR+1)
130     PRINT VALUE
```

WAIT Statement

Purpose:

Suspends execution of a program while monitoring the status of an I/O port.

Version:

LC4

Format:

WAIT port, n [,m]

Comments:

port is the I/O address in the range of 0 to 255.

n, m are integer expressions in the range of 0 to 255.

The WAIT statement suspends execution until a specified bit pattern appears at the specified I/O location. The data at the port is first XORed with the integer expression m and then ANDed with n. If the result is not zero, then program execution continues with the next statement. If the result is zero, BASIC loops back and reads the I/O location again. m is optional and if omitted, it is assumed to be zero.

Refer to the LC4 Hardware Description And Installation manual (Form 157) for a description of valid I/O locations.

WARNING!: It is possible to remain in an infinite loop when using the WAIT statement if a port is specified which never changes. To interrupt the loop and stop the program, use Ctrl-C.

Example:

This example waits for the third bit at I/O location 80 Hex to become a 1 (this can correspond to an input module in position 2 of a PAMUX station jumpered for address 0).

```
100     WAIT &H80,4          'Wait For Input Module 3 To Go High
```

WHILE And WEND Statements

Purpose:

Statements within the loop are executed as long as a given condition is true.

Version:

LC2, LC4

Format:

WHILE expression
.
.
(statements)
.
.
WEND

expression is any numeric expression.

Comments

If the expression is true (non-zero), the statements will be executed until a WEND statement is encountered. When the WEND statement is reached, LC2/LC4 BASIC will loop back to the WHILE statement and check the expression. If expression is still true, the process will be repeated. If expression is false, execution will resume with the statement following the WEND statement.

WHILE... WEND loops may be nested to fifteen levels. Each WEND will match the most recent WHILE. If LC2/LC4 BASIC does not find a matching WEND for a given WHILE, a "WHILE without WEND" error message is displayed. Similarly, an unmatched WEND statement will cause a "WEND without WHILE" error.

Example:

This example will continue to prompt for, and execute commands by calling appropriate subroutines until "Q" is entered in response to the prompt.

```
100     WHILE A$ <> "Q"
110     INPUT "ENTER COMMAND (Q to quit): ";A$
120     IF A$ = "D" GOSUB 1000
130     IF A$ = "E" GOSUB 2000
140     IF A$ = "C" GOSUB 3000
150     IF A$ = "S" GOSUB 4000
160     IF A$ = "F" GOSUB 5000
170     WEND
180     STOP
```

APPENDIX A

Error Messages

The LC2/LC4 BASIC is capable of detecting many errors which cause a program to stop running. Upon detection of such an error, an error message is displayed. The value shown as ERR represents the value that will be placed in the ERR system variable.

The following is a list of all the LC2/LC4 BASIC error messages:

NEXT Without FOR (ERR 1)

A NEXT statement has been encountered without a matching FOR statement. Make sure the variable used in the NEXT statement is the same as the variable specified in the FOR statement.

Syntax Error (ERR 2)

A program line contains an incorrect sequence of characters; such as, incorrect punctuation or a misspelled statement or command.

RETURN Without GOSUB (ERR 3)

A RETURN statement has been encountered in the program without a GOSUB statement being executed first.

Out Of Data (ERR 4)

A READ statement is trying to read more data than is in the DATA statements.

Illegal Function Call (ERR 5)

A parameter which is out of range is passed to a LC2/LC4 BASIC function.

Overflow (ERR 6)

The magnitude of a number is too large for LC2/LC4 BASIC to represent in its number system. If it is an integer value, try converting it to a single precision, real value.

Out Of Memory (ERR 7)

A program is too large to fit in the available memory space.

Unknown Line (ERR 8)

A reference to a line in a statement or command refers to a line which doesn't exist in the program.

Dimension Error (ERR 10)

A variable has been assigned a dimension which is outside the range of values acceptable by LC2/LC4 BASIC.

Division By Zero (ERR 11)

An expression contained a statement which attempted to divide by zero or raise zero to a negative power.

Error Messages (Continued)

Type Mismatch (ERR 13)

A string value appeared where a numeric value was expected, or a numeric value appeared where a string value was expected.

String Too Long (ERR 15)

The expression is concatenating a string which is greater than 255 characters. String variables are limited to a length of 255 characters.

Cannot Continue (ERR 17)

Trying to execute a CONT statement after a program has been edited or before a program has been run.

RESUME Without Error (ERR 20)

The program has encountered a RESUME statement without having trapped an error. The error trapping routine should only be entered when an error occurs or an ERROR statement is executed. You probably need to include a STOP or END statement before the error trapping routine to prevent the program from going to the error trapping code.

Missing Operand (ERR 22)

An expression contains an operator, such as + or AND, with no operand following it.

FOR Without NEXT (ERR 26)

A FOR statement is missing a matching NEXT statement. Make sure the variable in the FOR statement is the same as the variable specified in the NEXT statement.

WHILE Without WEND (ERR 29)

A WHILE statement in the program does not have a matching WEND statement.

WEND Without WHILE (ERR 30)

A WEND has been encountered before a matching WHILE has been executed.

Bad File Number (ERR 52)

A statement uses a file number of a file that is not open, or the file number is out of the range of possible file numbers specified at initialization. Or, the device name in the file specification is too long or invalid, or the filename was too long or invalid.

File Not Found (ERR 53)

A FILES, KILL, or OPEN references a file that does not exist on the RAM disk.

Bad File Mode (ERR 54)

Trying to execute an OPEN statement with a file mode other that APPEND, INPUT, or OUTPUT

Error Messages (Continued)

File Already Open (ERR 55)

Trying to open a file that is already opened or trying to KILL a file that is open.

Device I/O Error (ERR 57)

An error occurred on a device I/O operation. When receiving communications data, this error can occur from an overrun, framing, break, or parity errors.

RAM Disk Directory Is Full (ERR 59)

Trying to open more than the allowable number of filenames on the RAM disk.

RAM Disk Is Full (ERR 61)

All RAM disk storage space is allocated.

Input Past End (ERR 62)

This is an end of file error. An input statement is executed for a null (empty) file, or after all the data in a sequential file was already input.

Use the EOF function to detect the end of file to avoid this error.

This error also occurs if you try to read from a file that was opened for append or output.

Bad File Name (ERR 64)

An invalid form is used for the filename with the FILES, KILL, or OPEN statement.

Garbage In File At Line nnnn (ERR 66)

This message occurs when the program has been corrupted because of memory loss or POKEs into incorrect memory locations. The line number where a problem occurred is indicated by nnnnn.

Break (ERR 100)

The Ctrl-C key combination has been pressed during program execution to terminate the program.

Out Of Variable Storage Space (ERR 101)

Additional variables cannot be created because there is not enough RAM space available for another variable.

Right Parenthesis Expected (ERR 102)

The program is missing a right parenthesis. There must be an equal number of right and left parentheses in an expression or statement.

Error Messages(Continued)

Left Parenthesis Expected (ERR 103)

The program is missing a left parenthesis. There must be an equal number of right and left parentheses in an expression or statement.

Array Has Already Been Dimensioned (ERR 104)

The expression to dimension an array contains the name of a previously dimensioned array.

Gosub Stack Overflow (ERR 106)

Nesting GOSUB statements too deeply or missing RETURN statements. Make sure each subroutine is capable of returning to the GOSUB statement which called it.

FOR/NEXT Stack Overflow (ERR 107)

Too many FOR/NEXT loops for the available memory or branching out of a FOR/NEXT loop has caused the stack area to be exceeded.

String Stack Overflow (ERR 108)

Operations involving too many or exceedingly long strings have resulted in LC2/LC4's temporary string workspace to be used up. Breaking up string operations into several smaller parts may remedy this problem.

WHILE/WEND Stack Overflow (ERR 109)

Too many WHILE/WEND loops for the available memory or branching out of a WHILE/WEND loop has caused the stack area to be exceeded.

Input Conversion Error (ERR 111)

Data being entered into the system does not match the variable type assigned to receive the data.

APPENDIX B

ASCII Character Set

Decimal	Hex	Character		Decimal	Hex	Character
0	00	CTRL @	(NUL)	32	20	Space
1	01	CTRL A	(SOH)	33	21	!
2	02	CTRL B	(STX)	34	22	"
3	03	CTRL C	(ETX)	35	23	#
4	04	CTRL D	(EOT)	36	24	$
5	05	CTRL E	(ENQ)	37	25	%
6	06	CTRL F	(ACK)	38	26	&
7	07	CTRL G	(BEL)	39	27	'
8	08	CTRL H	(BS)	40	28	(
9	09	CTRL I	(HT)	41	29	)
10	0A	CTRL J	(LF)	42	2A	*
11	0B	CTRL K	(VT)	43	2B	+
12	0C	CTRL L	(FF)	44	2C	,
13	0D	CTRL M	(CR)	45	2D	-
14	0E	CTRL N	(SO)	46	2E	.
15	0F	CTRL O	(SI)	47	2F	/
16	10	CTRL P	(DLE)	48	30	0
17	11	CTRL Q	(DC1)	49	31	1
18	12	CTRL R	(DC2)	50	32	2
19	13	CTRL S	(DC3)	51	33	3
20	14	CTRL T	(DC4)	52	34	4
21	15	CTRL U	(NAK)	53	35	5
22	16	CTRL V	(SYN)	54	36	6
23	17	CTRL W	(ETB)	55	37	7
24	18	CTRL X	(CAN)	56	38	8
25	19	CTRL Y	(EM)	57	39	9
26	1A	CTRL Z	(SUB)	58	3A	:
27	1B	CTRL [	(ESC)	59	3B	;
28	1C	CTRL \	(FS)	60	3C	<
29	1D	CTRL]	(GS)	61	3D	=
30	1E	CTRL ^	(RS)	62	3E	>
31	1F	CTRL _	(US)	63	3F	?

ASCII Character Set (Continued)

Decimal	Hex	Character
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	\
93	5D	]
94	5E	^
95	5F	_

Decimal	Hex	Character
96	60	'
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	{
124	7C	|
125	7D	}
126	7E	~
127	7F	DEL

APPENDIX C

Multidropping Several LC2/LC4 On One Communications Line

In order to communicate or download different programs to each of several LC2/LC4's on a single communications link, the following steps should be performed.

There are two general procedures to be followed, assigning addresses and communicating with each local controller.

Assigning Addresses

Step 1:

Turn off power to all LC2/LC4's on the communications link.

Step 2:

LC2: Power up one of the LC2's and assign it a unique address by using the statement ADDRESS@ = n. Where n is the address you have selected. The statement is sent from the host terminal after the power up message is received and the NEW statement has been sent to clear LC2's memory. After the address is set, verify it by sending a PRINT ADDRESS@ statement. If all is well, turn off the LC2's power.

LC4: The LC4 has hardware address jumpers, refer to the LC4 Hardware Description And Installation manual on setting the address jumpers. After the address is set, turn on power and verify it by sending a PRINT ADDRESS@ statement. If all is well, turn off the LC4's power.

Repeat Step 2 for each LC2/LC4 on the link. When all LC2/LC4's have been assigned an address, continue with the next procedure.

Communicating With Each LC2/LC4

Step 1:

Power up all LC2/LC4's and make sure they are not currently running programs. (The AUTOBOOT jumper must not be installed, refer to the LC2 or LC4 Hardware Description And Installation manual.)

Step 2:

From the host terminal, issue a SLEEP command (with no address specified). This will cause all LC2/LC4's which are actively listening to become disabled.

Step 3:

From the host terminal, issue an AWAKE n command, where n is the address of the selected LC2/LC4 which is to receive a downloaded program.

Step 4:

Enter or download the appropriate program to the LC2/LC4. It should be the only unit listening and responding on the link.

Step 5:

After the program has been downloaded, send a SLEEP command to disable the LC2/LC4 which is awake.

Step 6:

Repeat steps 3, 4, and 5 until all LC2/LC4's on the link have been programmed.

Step 7:

From the terminal, issue an AWAKE command (with no address specified). This will enable all LC2/LC4's currently asleep. Now sending a RUN command will cause all LC2/LC4's to begin executing their programs.

APPENDIX D

Sample Programs

LC2/LC4 to IBM PC Communications with the LC2/LC4 Being Interrupted by the IBM PC

This example shows how the LC2/LC4 can be programmed to interrupt on characters sent by a host computer or terminal. The ON KEY statement is used to trap certain characters and jump to a specified subroutine which then executes a process. If the host is a computer, running a supervisory program, at selected times the host can interrupt LC2/LC4 to pass information back and forth by sending one character as the request command.

Program running in LC2/LC4:

```
10      ON KEY("A") GOSUB 100
20      ON KEY("B") GOSUB 200
30      ON KEY("C") GOSUB 300
40      KEY("A") ON
50      KEY("B") ON
60      KEY("C") ON
.
.       '*    MAIN PROGRAM
.
99      END
100     '*    ROUTINE TO PROCESS KEY A
110     '*
120     FOR I% = 1 TO 10                'Loop To Pass Parameters
130     PRINT X%(I%), Y%(I%)            'Send Host Two Arrays
140     NEXT                            'End The Loop
.
.
.
180     RETURN
.
200     '*    ROUTINE TO PROCESS KEY B
210     '*
.
.
280     RETURN
.
300     '*    ROUTINE TO PROCESS KEY C
310     '*
.
.
380     RETURN
```

LC2/LC4 to IBM PC Communications with the IBM PCbeing Interrupted by LC2/LC4

The following is an example of a LC2/LC4 device interrupting a host computer (in this case an IBM PC). The program in the host is running at the same time as the program in LC2/LC4. The example is not a complete program, but shows a method of alarm logging to a host.

Program in LC2/LC4:

```
1000    PRINT "ALARM HAS OCCURRED AT STATION 1"
.
.
2000    PRINT "ALARM HAS OCCURRED AT STATION 4"
.
.
3000    END
```

Program in IBM PC (running concurrent to LC2/LC4's program)

```
100     ON ERROR GOT0 2000                  'Retry On Errors
110     OPEN "COM1:19200,N,8,1" AS #1 '     Open Port To LC2/LC4
120     ON COM(1) GOSUB 1000                'Interrupt On Messages From LC2/LC4
130     COM(1) ON                           'Turn On Interrupts
.
.
            'MAIN PROGRAM
.
.
.
900     END
1000    FOR I% = 1 TO 100                   'Delay For Message Completion
1010    NEXT
1020    IF LOC(1) = 0 THEN RETURN           'Make Sure Buffer Is Not Empty
1030    MESSG$ = INPUT$(LOC(1),#1)          'Get Message
1040    LOCATE 1,1:PRINT MESSG$             'Display Message
1050    OPEN "ALARMS" FOR APPEND AS #2      'Open Alarm File
1060    PRINT #2,DATE$,TIME$,MESSG$         'Time Stamp And Save Message
1070    CLOSE #2                            'Close Alarm File
1080    RETURN                              'Return
2000    RESUME                              'If Error Then Retry
```

A Sample LC2/LC4 Basic Program

```
10    '*************************************************************************
20    '*
30    '*              DATA ACQUISITION (LC2/LC4 VERSION)
40    '*
50    '*   This program continuously reads the status of 3 boards,
60    '*   (2 discrete and 1 analog) and maintains a table of the
70    '*   current status.  After each read, a check is made between
80    '*   the current status and the previous status.  If a change
90    '*   is encountered, a message is constructed and sent to the
100   '*   host terminal or computer.  The message describes the
110   '*   event and also contains time and date information.  At
120   '*   any time, LC2/LC4 can be interrupted by a host terminal or
130   '*   computer which sends the 'S' character.  When LC2/LC4
140   '*   senses that a 'S' character is received, a complete message
150   '*   is sent to the host, which contains information on all the
160   '*   status points in the table.
170   '*
180   '*   Futhermore, a small control algorithm is implemented to
190   '*   activate outputs whenever counters on corresponding inputs
200   '*   reach a multiple of five counts.  An analog input is also
210   '*   used to set an analog output.
220   '*
230   '*   This program serves no actual useful purpose except to
240   '*   illustrate the concept of LC2/LC4 to HOST and HOST to
250   '*   LC2/LC4 interaction.
260   '*
270   '*
280   '*
290   '*************************************************************************
300   '*
310   '*************************************************************************
320   '*
330   '*              MAIN PROGRAM
340   '*
350   '*************************************************************************
360   GOSUB 4750            'Run The Initallization routines
370   '*
380   '*************************************************************************
390   '*
400   '*              PROCESS LOOP
410   '*
420   '*************************************************************************
430   GOSUB 1470            'Read The OPTOMUX Counters
440   GOSUB 1040            'Check Counters And Set Outputs Accordingly
450   GOSUB 1170            'Read Analog Inputs
460   GOSUB 1340            'Set Analog Output Based On Analog Input #3
470   GOSUB 1660            'Read Status Of Discrete Points
480   GOSUB 710             'Compare New Status To Old Status Arrays
490   GOTO 430              'Do The Process Loop Over Again
500   END                   'End Of Program
```

```
510     '*************************************************************************
520     '*
530     '*              SUBROUTINES
540     '*
550     '*************************************************************************
560     '*************************************************************************
570     '*
580     '*              ROUTINE THAT CALLS THE OPTOWARE DRIVER
590     '*
600     '*************************************************************************
610     '*
620     '*
630     CALL OPTOWARE (ERRCOD%,ADDR%,CMD%,POSIT%(0),MODIF%(0),INFO%(0))
640     IF ERRCOD% < 0 THEN GOSUB 3430
650     RETURN
660     '*************************************************************************
670     '*
680     '*    PROCEDURE WHICH COMPARES NEW VALUES TO OLD VALUES
690     '*
700     '*************************************************************************
710     '*
720     FOR I% = 1 TO 4
730     IF B%(I%) <> A%(I%) THEN GOSUB 2160  'If Different, Display Message
740     NEXT
750     FOR I% = 5 TO 7
760     '*
770     '*    IF ANALOG VALUES OUTSIDE DEADBAND OF OLD VALUES, DISPLAY MESSAGE
780     '*
790     IF (B%(I%)< A%(I%) - DBND/2) OR (B%(I%)> A%(I%)+DBND/2) THEN GOSUB 2160
800     NEXT
810     FOR I% = 8 TO 11
820     '*
830     '*    IF COUNTERS ARE DIFFERENT, DISPLAY MESSAGE
840     '*
850     IF B%(I%) <> A%(I%) THEN GOSUB 2160
860     NEXT
870     GOSUB 940              'Update Old Array To Equal New Array
880     RETURN
890     '*************************************************************************
900     '*
910     '*              SETS OLD ARRAY EQUAL TO NEW ARRAY
920     '*
930     '*************************************************************************
940     '*
950     FOR I% = 1 TO 11
960     B%(I%) = A%(I%)
970     NEXT
980     RETURN
990     '*************************************************************************
1000    '*
1010    '*    CHECKS THE COUNTER VALUES AND SETS APPROPRIATE OUTPUTS
1020    '*
1030    '*************************************************************************
1040    '*
1050    FOR I% = 8 TO 11
1060    '*
1070    '*    IF MULTIPLE OF 5, TURN ON OUTPUT ELSE TURN OFF OUTPUT
1080    '*
1090    IF (A%(I%) MOD 5) = 0 THEN GOSUB 1820 ELSE GOSUB 1940
1100    NEXT
1110    RETURN
1120    '*************************************************************************
1130    '*
1140    '*    READ ANALOG INPUTS
1150    '*
1160    '*************************************************************************
1170    '*
1180    ADDR% = BOARD3
1190    CMD% = 37                      'Read Analog Inputs
1200    POSIT%(0) = 0
```

```
1210    POSIT%(1) = 1
1220    POSIT%(2) = 2
1230    POSIT%(3) = -1                  'End Of The List
1240    GOSUB 610                       'Call The Driver
1250    A%(5) = INFO%(0)
1260    A%(6) = INFO%(1)
1270    A%(7) = INFO%(2)
1280    RETURN
1290    '*************************************************************************
1300    '*
1310    '*    UPDATES THE ANALOG OUTPUT
1320    '*
1330    '*************************************************************************
1340    '*
1350    ADDR% = BOARD3
1360    CMD% = 35                       'Write Analog Output
1370    POSIT%(0) = M1
1380    POSIT%(1) = -1
1390    INFO%(0) = A%(7)
1400    GOSUB 610                       'Call The Driver
1410    RETURN
1420    '*************************************************************************
1430    '*
1440    '*    READ ALL THE COUNTERS
1450    '*
1460    '*************************************************************************
1470    '*
1480    ADDR% = BOARD1
1490    CMD% = 22
1500    POSIT%(0) = S1
1510    POSIT%(1) = S2
1520    POSIT%(2) = -1
1530    GOSUB 610                       'Call The Driver
1540    A%(8) = INFO%(0)
1550    A%(9) = INFO%(1)
1560    ADDR% = BOARD2
1570    GOSUB 610                       'Call The Driver
1580    A%(10) = INFO%(0)
1590    A%(11) = INFO%(1)
1600    RETURN
1610    '*************************************************************************
1620    '*
1630    '*              READ DIGITAL INPUT STATUS
1640    '*
1650    '*************************************************************************
1660    '*
1670    ADDR% = BOARD1
1680    CMD% = 12                       'Read Status Command
1690    GOSUB 610                       'Call The Driver
1700    A%(1) = INFO%(0)
1710    A%(2) = INFO%(1)
1720    ADDR% = BOARD2
1730    GOSUB 610                       'Call The Driver
1740    A%(3) = INFO%(0)
1750    A%(4) = INFO%(1)
1760    RETURN
1770    '*************************************************************************
1780    '*
1790    '*    TURNS ON A SPECIFIC OUTPUT
1800    '*
1810    '*************************************************************************
1820    '*
1830    IF I% < 10 THEN ADDR% = BOARD1 ELSE ADDR% = BOARD2
1840    CMD% = 10
1850    IF (I% = 8) OR (I% = 10) THEN POSIT%(0) = 2 ELSE POSIT%(0) = 3
1860    POSIT%(1) = -1
1870    GOSUB 610                       'Call The Driver
1880    RETURN
1890    '*************************************************************************
1900    '*
```

```
1910    '*    TURNS OFF A SPECIFIC OUTPUT
1920    '*
1930    '*************************************************************************
1940    '*
1950    IF I% < 10 THEN ADDR% = BOARD1 ELSE ADDR% = BOARD2
1960    CMD% = 11                       'Turns Off Outputs
1970    IF (I% = 8) OR (I% = 10) THEN POSIT%(0) = 2 ELSE POSIT%(0) = 3
1980    POSIT%(1) = -1
1990    GOSUB 610                       'Call The Driver
2000    RETURN
2010    '*************************************************************************
2020    '*
2030    '*    CONVERT ANALOG VALUES TO ENGINEERING UNITS
2040    '*
2050    '*************************************************************************
2060    '*
2070    TEMP1 = (A%(5)*.08262) - 188.4          'Convert Temp Sensor To Degrees C
2080    TEMP2 = (A%(6)*.08262) - 188.4          'Convert Temp Sensor To Degress C
2090    VOLTS = (5/4096) * A%(7)                'Convert 5 Volt Reading To Volts
2100    RETURN
2110    '*************************************************************************
2120    '*
2130    '*              SENDS STATUS CHANGE MESSAGE TO HOST
2140    '*
2150    '*************************************************************************
2160    '*
2170    GOSUB 2060                              'Calculate Analog Engineering Units
2180    GOSUB 2260                              'English Status
2190    PRINT DATE$;" ";TIME$;" ";ITEM$(I%);" ";STATUS$(I%);" ";UNITS$(I%)
2200    RETURN
2210    '*************************************************************************
2220    '*
2230    '*    SETS THE STATUS MESSAGE VARIABLES TO ENGLISH EQUIVALENTS
2240    '*
2250    '*************************************************************************
2260    '*
2270    FOR J% = 1 TO 4
2280    IF A%(J%) = 1 THEN STATUS$(J%) = "ON " ELSE STATUS$(J%)="OFF"
2290    NEXT
2300    STATUS$(5) = STR$(TEMP1)
2310    STATUS$(6) = STR$(TEMP2)
2320    STATUS$(7) = STR$(VOLTS)
2330    FOR J% = 8 TO 11
2340    STATUS$(J%) = STR$(A%(J%))
2350    NEXT
2360    RETURN
2370    '*************************************************************************
2380    '*
2390    '*    INTERRUPT SERVICE ROUTINE FOR TRAPPED KEYS
2400    '*
2410    '*************************************************************************
2420    '*
2430    GOSUB 2060                              'Calculate Engineering Units For Analog
2440    GOSUB 2260                              'Update Status Messages In English
2450    PRINT
2460    PRINT "                 *** CURRENT SYSTEM STATUS ***"
2470    PRINT
2480    PRINT "                         TODAY'S DATE: ";DATE%;"   TIME: ";TIME$
2490    PRINT
2500    FOR M% = 1 TO 11
2510    GOSUB 2610                              'Display Message For Each I/O Point
2520    NEXT
2530    PRINT
2540    PRINT
2550    RETURN
2560    '*************************************************************************
2570    '*
2580    '*    BUILD AND SEND MESSAGE TO HOST
2590    '*
2600    '*************************************************************************
```

```
2610    '*
2620    PRINT ITEM$(M%);" " ;STATUS$(M%);" ";UNITS$(M%)
2630    RETURN
2640    '*    System Constants
2650    OPTOWARE = 4                    'Location Of The OPTOWARE Driver
2660    DIM POSIT%(15)                  'Dimension POSITIONS Array
2670    DIM INFO%(15)                   'Dimension INFO Array
2680    DIM MODIF%(1)                   'Dimension MODIFIERS Array
2690    DIM A%(15)                      'Dimension New Status Array
2700    DIM B%(15)                      'Dimension Old Status Array
2710    DIM ITEM$(15)                   'Message Array For Item Description
2720    DIM UNITS$(15)                  'Message Array For Engineering Units
2730    DIM STATUS$(15)                 'Array To Represent Status In English
2740    '*
2750    '*    CONSTANTS WHICH SET UP ADDRESSES AND POSITIONS OF OPTOMUX I/O
2760    '*
2770    BOARD1 = 255                    'Address Of Board #1 (Digital I/O)
2780    BOARD2 = 254                    'Address Of Board #2 (Digital I/O)
2790    BOARD3 = 239                    'Address Of Board #3 (Analog I/O)
2800    L1 = 2                          'Output #1, Board 1, Position 2
2810    L2 = 3                          'Output #2, Board 1, Position 3
2820    L3 = 2                          'Output #3, Board 2, Position 2
2830    L4 = 3                          'Output #4, Board 2, Position 3
2840    S1 = 0                          'Input #1, Board 1, Position 0
2850    S2 = 1                          'Input #2, Board 1, Position 1
2860    S3 = 0                          'Input #3, Board 2, Position 0
2870    S4 = 1                          'Input #4, Board 2, Position 1
2880    T1 = 0                          'Analog Input #1, Board 3, Position 0
2890    T2 = 1                          'Analog Input #2, Board 3, Position 1
2900    R1 = 2                          'Analog Input #3, Board 3, Position 2
2910    M1 = 3                          'Analog Output #1, Board 3, Position 3
2920    '*
2930    '*    DEADBAND FOR COMPARING ANALOG VALUES
2940    '*
2950    DBND = 10                       'Deadband Value
2960    '*
2970    '*    MESSAGE CONSTANTS USED FOR DESCRIBING POINTS AND EVENTS
2980    '*
2990    ITEM$(1) = "SWITCH S1           "
3000    ITEM$(2) = "SWITCH S2           "
3010    ITEM$(3) = "SWITCH S3           "
3020    ITEM$(4) = "SWITCH S4           "
3030    ITEM$(5) = "TEMPERATURE PROBE #1"
3040    ITEM$(6) = "TEMPERATURE PROBE #2"
3050    ITEM$(7) = "POTENTIOMETER       "
3060    ITEM$(8) = "COUNTER FOR S1      "
3070    ITEM$(9) = "COUNTER FOR S2      "
3080    ITEM$(10)= "COUNTER FOR S3      "
3090    ITEM$(11)= "COUNTER FOR S4      "
3100    UNITS$(1) = "           "
3110    UNITS$(2) = "           "
3120    UNITS$(3) = "           "
3130    UNITS$(4) = "           "
3140    UNITS$(5) = "DEGREES C          "
3150    UNITS$(6) = "DEGREES C          "
3160    UNITS$(7) = "VOLTS      "
3170    UNITS$(8) = "COUNTS     "
3180    UNITS$(9) = "COUNTS     "
3190    UNITS$(10)= "COUNTS     "
3200    UNITS$(11)= "COUNTS     "
3210    RETURN
3220    '*************************************************************************
3230    '*
3240    '*                INITIALIZE OPTOMUX VARIABLES
3250    '*
3260    '*************************************************************************
3270    '*
3280    FOR I% = 0 TO 15
3290    POSIT%(I%) = 0
3300    INFO%(I%) = 0
```

```
3310    NEXT
3320    MODIF%(0) = 0
3330    MODIF%(1) = 0
3340    ADDR% = 0
3350    ERRCOD% = 0
3360    CMD% = 0
3370    RETURN
3380    '*************************************************************************
3390    '*
3400    '*              ERROR CHECK ROUTINE
3410    '*
3420    '*************************************************************************
3430    '*
3440    PRINT"*************************************************************"
3450    PRINT "       OPTOMUX COMMUNICATIONS ERROR"
3460    PRINT DATE$;" ";TIME$;" ERROR#: ";ERRCOD%;" AT ADDRESS: ";ADDR%
3470    PRINT  "*************************************************************"
3480    RETURN
3490    '*************************************************************************
3500    '*
3510    '*              ROUTINE TO SEND POWER UP CLEAR TO BOARD 1
3520    '*
3530    '*************************************************************************
3540    '*
3550    ADDR% = BOARD1
3560    CMD% = 0
3570    GOSUB 610                       'Call The Driver
3580    RETURN
3590    '*
3600    '*************************************************************************
3610    '*
3620    '*              ROUTINE TO SEND POWER UP CLEAR TO BOARD 2
3630    '*
3640    '*************************************************************************
3650    '*
3660    ADDR% = BOARD2
3670    CMD% = 0
3680    GOSUB 610                       'Call The Driver
3690    RETURN
3700    '*
3710    '*************************************************************************
3720    '*
3730    '*    ROUTINE TO SEND POWER UP CLEAR TO BOARD 3
3740    '*
3750    '*************************************************************************
3760    '*
3770    ADDR% = BOARD3
3780    CMD% = 0
3790    GOSUB 610                       'Call The Driver
3800    RETURN
3810    '*************************************************************************
3820    '*
3830    '*    ROUTINE TO RESET BOARD 1
3840    '*
3850    '*************************************************************************
3860    '*
3870    ADDR% = BOARD1
3880    CMD% = 1
3890    GOSUB 610                       'Call The Driver
3900    RETURN
3910    '*************************************************************************
3920    '*
3930    '*    ROUTINE TO RESET BOARD 2
3940    '*
3950    '*************************************************************************
3960    '*
3970    ADDR% = BOARD2
3980    CMD% = 1
3990    GOSUB 610                       'Call The Driver
4000    RETURN
```

```
4010  '*************************************************************
4020  '*
4030  '*    ROUTINE TO RESET BOARD 3
4040  '*
4050  '*************************************************************
4060  '*
4070  ADDR% = BOARD3
4080  CMD% = 1
4090  GOSUB 610                    'Call The Driver
4100  RETURN
4110  '*************************************************************
4120  '*
4130  '*    ROUTINE TO CONFIGURE OUTPUTS
4140  '*
4150  '*************************************************************
4160  '*
4170  CMD% = 8
4180  ADDR% = BOARD1
4190  POSIT%(0) = L1               'Also L3
4200  POSIT%(1) = L2               'Also L4
4210  POSIT%(2) = -1               'End The List
4220  GOSUB 610                    'Call The Driver
4230  ADDR% = BOARD2
4240  GOSUB 610                    'Call The Driver
4250  ADDR% = BOARD3
4260  POSIT%(0) = M1
4270  FOR I% = 4 TO 15
4280  POSIT%(I% - 3) = I%
4290  NEXT
4300  POSIT%(13) = -1              'End The List
4310  GOSUB 610                    'Call The Driver
4320  RETURN
4330  '*************************************************************
4340  '*
4350  '*    INITIALIZE THE DATA ARRAYS
4360  '*
4370  '*************************************************************
4380  '*
4390  FOR I% = 1 TO 15
4400  A%(I%) = 0
4410  B%(I%) = 0
4420  NEXT
4430  RETURN
4440  '*************************************************************
4450  '*
4460  '*          SET UP THE KEYS TO TRAP FROM HOST
4470  '*
4480  '*************************************************************
4490  '*
4500  ON KEY("S") GOSUB 2420       'Goto Interrupt Service Routine On A Trap
4510  ON KEY("s") GOSUB 2420       'Goto Interrupt Service Routine On A Trap
4520  KEY("S") ON                  'Enable Key Trapping
4530  KEY("s") ON                  'Enable Key Trapping
4540  RETURN
4550  '*************************************************************
4560  '*
4570  '*    STARTS COUNTERS ON ALL SWITCHES
4580  '*
4590  '*************************************************************
4600  '*
```

```
4610    ADDR% = BOARD1
4620    CMD% = 19
4630    POSIT%(0) = S1
4640    POSIT%(1) = S2
4650    POSIT%(2) = -1
4660    GOSUB 610                   'Call The Driver
4670    ADDR% = BOARD2
4680    GOSUB 610                   'Call The Driver
4690    RETURN
4700    '*************************************************************************
4710    '*
4720    '*    INITIALIZATION ROUTINES
4730    '*
4740    '*************************************************************************
4750    '*
4760     GOSUB 2640                 'Initialize Variables
4770     GOSUB 3270                 'Initialize OPTOMUX Arrays To Zero
4780     GOSUB 3540                 'Send Power Up Clear To Board 1
4790     GOSUB 3650                 'Send Power Up Clear To Board 2
4800     GOSUB 3760                 'Send Power Up Clear To Board 3
4810     GOSUB 3860                 'Reset Board 1
4820     GOSUB 3960                 'Reset Board 2
4830     GOSUB 4060                 'Reset Board 3
4840     GOSUB 4160                 'Configure Output Positions On All Boards
4850     GOSUB 4380                 'Initialize Status Arrays A And B
4860     GOSUB 4600                 'Start Counters On All Digital Inputs
4870     GOSUB 4490                 'Set Up Key Trapping Routine
4880     RETURN
```

APPENDIX E

Tips and Techniques

For best readability, programs should be written in a modular fashion with lots of comments. The term "modular" implies the use of subroutines accessed with the GOSUB statement. The GOTO statements should not be used if it can be helped. Each subroutine should perform only one function.

For maximum speed, place the critical subroutines which are called the most often, at the beginning of the program. This reduces the time LC2/LC4's BASIC spends in searching the program for the correct line number when a GOSUB statement is executed. Comments should be removed if speed optimization is necessary. For maximum performance but less readability, do not use subroutines, but rather place all code in line and repeat it where needed.

Preset the variables to be used in the CALL OPTOWARE statement early in the program. Then use different OPTOWARE CALLS with different sets of parameters so as not to reassign variables during the program's execution. Remember, assignment costs time.

If comments are to be removed in order to make room for more program code or for speed optimization, then a commented file or a commented listing should be kept for documentation and debugging purposes.

Use the ON TIMER function for time dependent tasks.

Routines which initialize the program or devices and are only performed at the start of the program should be placed at the end of the program and should be the first thing called from the main program loop.

Constants, such as the addresses of OPTOMUX stations should still be assigned to variables in case their value should need to be changed at a later time. This makes the program more readable and later allows a change to be made in one place rather than at every place in the program where that parameter is used.

NOTE: LC2/LC4 BASIC allows the same program to be written in a variety of ways. Some ways are better than others and the method is dependent on the application. What works great in one application may not work well in the next. You might even need to experiment to find the best solution.

The NEW command must be issued prior to entering or downloading a program.

APPENDIX F

Exceptions & Differences from IBM/Microsoft BASIC

The following is a list of differences between LC2/LC4 BASIC and IBM/Microsoft BASIC. Refer to the section "Commands Not Supported in LC2/LC4 BASIC" for information on specific commands which are not supported.

The following apply to LC2/LC4 BASIC:

- Arrays are one dimension and up to 255 elements
- Floating point numbers are single-precision numbers.
- Only one statement or command is allowed per program line.
- Commas or semicolons must be used as delimiters for commands, spaces are not allowed as delimiters.
- The CLEAR command with the option will reserve space at the beginning of memory and does not reserve stack space.
- The STR$ function will return a leading space on a fractional number.
- There is no "line" option on the RETURN command.

APPENDIX G

IBM/Microsoft BASIC Commands Not Supported in LC2/LC4 BASIC

ATN	DEFSTR	OPEN	SGN
AUTO	DEFUSR	OPTION BASE	SOUND
BEEP	DRAW	PAINT	SPC(n)
BLOAD	EDIT	PEN	STICK
BSAVE	ERASE	PEN OFF	STRIG(n)
CDBL	FIELD	PEN ON	STRIG(n) OFF
CHAIN	GET	PEN STOP	STRIG(n) ON
CIRCLE	LOAD	PLAY	STRIG(n) STOP
CLS	LOCATE	POINT	STRIG OFF
COLOR	LPOS	POS	STRIG ON
COMMON	LPRINT USING	PRESET	STRING$
CSNG	LSET	PRINT #f USING	TAB
CSRLIN	MERGE	PSET	WIDTH
CVD	MKD$	PUT	WRITE
CVI	MKI$	RANDOMIZE	WRITE #f
CVS	MKS$	RENUM	TAN
DEFDBL	MOTOR	RESET	USRN(x)
DEFFN	NAME	RND	VARPTR$
DEFINT	OCT$	RSET	VARPTR (#f)
DEFSEG	ON PEN GOSUB	SAVE	
DEFSNG	ON STRIG GOSUB	SCREEN	

OPTO 22

43044 Business Park Drive • Temecula, CA 92590-3614
Phone: 800/321-OPTO (6786) or 909/695-3000
Fax: 800/832-OPTO (6786) or 909/695-2712
Internet Web site: http://www.opto22.com

Product Support Services:
800/TEK-OPTO (835-6786) or 909/695-3080
Fax: 909/695-3017
E-mail: support@opto22.com
Bulletin Board System (BBS): 909/695-1367
FTP site: ftp.opto22.com