Chapter 1 Introduction

The NCMZ-1799 is a general purpose communications module. Mounted in an I/O rack, the 1799 allows the PC1000 family of Programmable Controllers to communicate with a variety of devices. Features include:

- Adjustable baud 300 - 38400 BPS
- Asynchronous 7,8,9,10,11 or 12 bit frame
- Dual Port access to PC1000 I/O bus allows fast CPU access
- 3 communication channels RS232
- 1 communication channel RS485
- fully buffered interrupt driven serial ports
- 32k bytes RAM memory, 26k bytes free for user program
- 16k bytes EPROM available (12k bytes if certain option packages installed)
- string variables and string functions
- multiple statements per line
- multitasking capabilities
- battery backed real time clock
- program back-up in EPROM
- high speed of execution
- access to most PC I/O registers (OR, OG, IR, IG) without any ladder program being required
- extensive program-editing facilities
- optional external 5 Vdc power supply can be used to reduce CPU power drain
- rack position independent
- user written assembly language subroutines can be mixed with BASIC
- programs can be locked, preventing modification or examination
- can be used as a remote I/O controller using PC1 100's as remote I/O drops

Various firmware options may be installed to further modify its operation. Refer to the Westinghouse Programmable Controller Price List 16-360 for more information.

This manual describes the BASIC interpreter and Modbus Slave option (see page 103). This version of the module must be programmed by the user before it will perform any function. You may write your own programs in a variety of languages including BASIC, 6809 assembly or C. Support for the BASIC language is built in. You must buy an assembler or C compiler if you wish to program in either of these two languages. This manual supplies nearly all the information you will need to program this module in BASIC.

To program in assembly or C, you will need to refer to the schematic diagrams in Appendix R. Additionally, if you plan on programming any serial port functions from C or assembly, you will need the Motorola 68681 DUART manual. This manual can be ordered directly from Motorola.

Although programming in C or other high level language may add many significant features (floating point math, etc.), the BASIC supplied with this module is optimized to quite easily handle most communications oriented tasks.

Since this module may be used for a variety of purposes, Westinghouse cannot assume any responsibility for any applications or uses of this program, nor for any errors or the consequent results there of.

1.1 Beginners All Purpose Symbolic Instruction Code (BASIC)

Several excellent books exist that teach the fundamentals of BASIC. You are encouraged to do further reading on this programming language. However, there are a few rules that will help in your understanding of this important language.

Line Numbers
All BASIC statements begin with a line number.

```
100   PRINT 'This is a test'
110   PRINT "This is another test"
```

The BASIC interpreter automatically sons the program so that statements with a lower line number appear in the listing before the statements with a higher line number.

Equations
BASIC is particularly adept at certain math functions.

```
100   A = B*lOO                multiply the variable B by 100 and place result in variable A
110   B = (C/100)+(45*D)       combination of division, addition and multiplication
```

Variables
Data is stored by BASIC in memory locations. These memory locations are accessed by your program. These memory locations are called 'variables", because they are referenced in your program by an algebraic variable (like A, B, etc.). BASIC offers several types of variables,

Integer Variables
Store numbers ranging from -32768 to +32767 (optionally, 0 to 65535). BASIC reserves two bytes (16 bits) of memory for this variable type. The NCMZ BASIC interpreter calls this type of variable, global.

```
Example :   A = -5000
            A = $C100      (store the value in hexadecimal)
```

Byte Variables
Store numbers ranging from 0 to 255. The NCMZ BASIC interpreter calls this type of variable local.

```
Example :   LA = 100
```

String Variables
Stores a sequence of bytes that are not interpreted as a number. Typically used to store messages, binary protocols and other sequence of characters that have meaning only when viewed as a combination of characters.

```
Example :   A$ = 'This is a test message'
```

Array Variables
Stores a whole table of numbers. The NCMZ BASIC interpreter will permit you to designate one or more of the integer variables to be array variables.

```
Example :   A(20) = -3421
            A(21) = 1110
            A(22) = 7
```

Interpreter

The microprocessor used by the NCMZ module only understands the native instruction code for that chip. The native code is call 'machine code". Machine code (or its cousin assembly language') is hard to read and worse, requires many instructions just to perform a simple task.

One way of working around those disadvantages is to write a special program for the microprocessor that would read a block of ASCII statements and expand each statement into appropriate machine code. The ASCII statements would be formatted in such a way that they would be easy to read, far easier than the equivalent assembly code.

For example, the IBM PC microprocessor (8088) could be programmed to take two numbers from memory, add them together and then place the result somewhere else in memory. The assembly code to do that would look like:

```
PUSH AX            ;save AX register
MOV  AX,Variable1  ;put Variable1 into microprocessor's AX register
ADD  AX,Variable2  ;add Variable2 to AX register
MOV  Variable3,AX  ;put sum back into memory
POP  AX            ;restore AX register
```

In BASIC the same task would look like:

```
100    C = A + B    :REM add A and C together, place result in C
```

Certainly the BASIC code is much easier to read than the assembly code. But remember, the microprocessor is running an "interpreter' that interprets (converts) the BASIC code into the native machine code for that particular microprocessor. The interpreter must be told to convert the BASIC program (typed in ASCII) into machine code. This is accomplished by typing:

```
RUN
```

at the command prompt. 'RUN' tells the interpreter to convert, on a line by line basis, each BASIC statement into the appropriate microprocessor machine code.

Once your program is running, the prompt disappears and the interpreter begins executing your program.

It may be a little confusing just how you send a 'RUN" command to the module. Well, the module is preprogrammed to accept ASCII characters on PORT A as direct command to the module. You could think of PORT A as being the 1799 program loader port.

The easiest way of sending ASCII characters to the module is with a terminal program running on an IBM PC (or any other type of computer for that matter). Westinghouse supplies a terminal program "NC799COM.EXE" that may be used for that purpose. Any other program (Relay, Smartcom, Crosstalk, etc.) would be just as good.

The 1799 module receives the characters sent from the terminal program and echoes each one back. In this way, the user has feedback that each character is being received properly. (Be sure and turn off the terminal "local echo' otherwise you will get two characters on the screen FFOORR EEWEERRYY OONNEE YYOOUU TTYYPPEE!!)

---

[1] For more information on 6809 assembly language programing, refer to the book "Programming the 6809" by Rodnay Zaks and William Labiak, Sybex, Inc. 2021 Challenger Drive, #100, Alameda, CA 94501. (415) 523-8233 or (800) 227-2346. Telex 336311. Library of Congress Card Number 82-50621, ISBN o-89588-078-4.

So, the IBM PC runs a terminal program that sends ASCII characters to the 1799, and the 1799 sends characters back to the terminal program. If you are using NC799COM.EXE the connection between the 1799 module PORT A and the computer is via COM1. Refer to Appendix G for pinouts of the proper cable to use.

You may be wondering if there is a way to stop the execution of your program. Perhaps your program isn't working properly and you want to make a change to the code. Or perhaps it is working, but you want to save the program to disk (you can only SAVE, LOAD and VERIFY while the module is in the PROMPT mode, not the RUN mode).

Well, there are three ways of stopping the module. One way is to have the program stop itself (see the END statement). Another way is if the module executes a statement that causes an internal error (divide by 0, overflow, invalid character, etc.). The third way is to send a special character to the 1799. That special character is called the c BREAK> character. A <BREAK> character is NOT the same thing as pressing <CTRL>-<BREAK> key sequence on the IBM PC keyboard. Rather, the <BREAK> is generated by the terminal program. (Crosstalk uses the END key, NC799COM uses the <CTRL>-C).

After the 1799 module receives this key sequence, it returns to the PROMPT mode and prints

READY
#

Modifying a Program

From the READY prompt, you can read and change variables.

#PRINT A,B,C	(prints the value of A, B and C)
#A=7	(change A to 7)
#PRHEX A(100)	(print the hexadecimal value of A(lOO))
#100 PRINT 'Hello'	(make line 100. If line 100 didn't exist, it is created. If line 100 existed, then this new line replaces it)

Clearing Memory

#NEW	(erase all memory)

Displaying Program

#LIST	(display all program statements)

1.2 Writing your first program

(Enter all commands and statements in UPPER CASE.)

<CTRL>C (Press the <CTRL>C key sequence)

#NEW

```
READY
#100 INPUT 'ENTER YOUR FIRST NAME ";A$
#110 INPUT "ENTER YOUR LAST NAME ";B$
#120 PRINT
#130 PRINT 'HELLO ';A$;' ";B$
#140 PRINT
#150 PRINT 'MY NAME IS NCMZ-1799
#160 END
```

#RUN

Type this program into your 1799 and see what it does. Experiment by changing some of the lines in the program. If you are using the NC799COM.EXE terminal program, save your program to disk by pressing the <F8> key, entering TEST, and then entering the drive letter you want to save the file to (for drive 'A" type A not A:). If you mistype the filename, press the <CTRL> C key.

If you make a typing error, just retype that line again. Any new line that has the same linenumber as a program statement already in memory will replace that old program statement.

Type NEW to totally erase a program from memory.

Type LIST to view the entire program.

Type LIST 110 to view line 110 only.

Type LIST 11 O-l 30 to view lines 110 through 130, inclusive.

Always NEW memory before loading a program from disk. If you don't the new program will not completely erase the old program. It will, however, overwrite program statements that have the same line number. If your new program uses line numbers that are completely different from the line numbers already in memory, the new program will 'append" to the old one and both programs will coexist.

1.3 Multitasking

You may have up to 4 programs running simultaneously in the same 1799.

These programs may be totally separate, or they may share data with one another. Data is shared between programs using the 'Global" variables (see Chapter 5. BASIC Variables).

```
#NEW

READY
#100 TASK 1000
#110 INPUT 'ENTER FIRST NUMBER ';A
#120 INPUT 'ENTER SECOND NUMBER ";B
#130 PRINT 'SUM = ";A+B
#140 GOT0 110
#1000 LEDON
#1010 SUSPEND 100
#1020 LEDOFF
#1030 SUSPEND 100
#1040 GOT0 1000
#RUN
```

This program prompts for two integers (+32767 to -32768 range) to be entered. It will add these two numbers together and print the answer. Meanwhile, a second program will flash LED 1 on and off continuously as long as the program is running. If the first program crashes due to an error (try entering 70000 as one of the numbers... you get an overflow error), the light stops flashing.

For more information on the TASK statement see page 64. Also see SUSPEND (page 63).