PART III – PROGRAMMING CONSIDERATIONS

3.1 MODEL 52 BASIC

It is not within the scope of this document to instruct the user in the fundamentals of general BASIC programming.

The INTEL 8052 microprocessor comes with its own BASIC interpreter coded in ROM on the chip. On chip space limitations require that some features of popular BASIC interpreters have been sacrificed. As such, the 8052 interpreter, referred to here as "BASIC-52" lacks a number of the commands common to other BASICs. Most functions provided by these commands can be duplicated, but more instructions are required to do so.

There are no graphics or screen control capabilities other than control characters that can be sent to the console.

This document provides a summary list of the BASIC-52 instruction set, but for a detailed reference on BASIC-52, the programmer will need Intel's MCS BASIC-52 USERS MANUAL, available from Intel or Compumotor. Order Compumotor part number 88-005680-01 or from Intel, publication #270010-002.

The Model 52 has twelve instructions that have been added to the BASIC-52 language to facilitate I/O operations, handling numerical strings, and program management. These are itemized in Sections 4.2 and 4.4 below.

3.1.1 VARIABLES

BASIC-52 has three kinds of variables. These are scalar, one dimension arrays, and string variables. Memory space must be reserved in advance for strings and for arrays that exceed 10 elements, using the STRING and DIM instructions.

Both scalar and array variable names may be up to eight letters and numbers in length. The programmer must take care that these names do not incorporate any of BASIC-52's reserved words.

Program execution is faster if variable names are restricted to one letter or a letter followed by a number as follows:

A, B, C,...,Z and A0, A1,...,A9,...,Z0,...,Z9

These variables are always "real" floating point numbers. For variables with values less than 65,536 (2 to the 16th power), it is also possible to do logical operations on them such as AND, OR, XOR, and NOT.

String variables all take the form "$(n)" where n must be a number between 0 and 8. String variables are limited to a fixed length which is declared in the STRING statement. String variables always start with variable "$(0)" and count up to the limit of declared space. The STRING statement also reserves the total amount of memory to be used by strings variables, thereby limiting the number. String space management requires an extra byte for each string variable, plus one overall.

Example: STRING 100, 10

reserves just enough memory for the 9 string variables, $(0) through $(8), 10 characters each.

References:

Variables:	BASIC-52 USERS MANUAL - Chapter 1.4
Strings:	BASIC-52 USERS MANUAL - Chapter 6
STRING:	BASIC-52 USERS MANUAL - Chapter 4.3

3.1.2 MANIPULATING CHARACTER STRINGS

Handling character strings in BASIC-52 is quite different. The following common string handling commands are not available in BASIC-52.

MID$, LEFT$, RIGHT$, STRING$

All of the above functions commands may be implemented with the BASIC-52 "ASC" and "CHR" commands. These two commands can incorporate a charcters pointer, and therefore operate very much like the MID$ command, in addition to duplicting the function of the regular ASC and CHR$ commands common to most BASICs.

BASIC-52 alone doesn't support the usual STR$ and VAL functions, but these have been added to the B52 instruction set as STR and VAL to facilitate RS232 transmission of numerical values.

String Examples:

The following instructions would display the ASCII code for the Nth character string variable A$ in Microsoft BASIC and from string variable *$(1)* in BASIC-52:

```
Microsoft: PRINT  ASC(MID$(A$,1,N))
8052: PRINT  ASC($(1),N)
```

The following instructions would display the Nth character from the string variables:

```
Microsoft: PRINT  MID$(A$,1,N)
8052: PRINT  CHR(ASC($(1),N),1)
```

References:

Strings:	BASIC-52 USERS MANUAL: Chapter 6
STR and VAL:	Section 4.2: Custom Instructions

3.1.3 INSTRUCTION ANOMALIES

BASIC-52 has limited capacity for manipulation of stored programs. The following common program control commands have no counterpart in BASIC-52.

```
MERGE, COMMON, KILL, NAME, EDIT, AUTO, RENUM,
       TRON, TROFF
```

There is no data transfer between programs.

Note that BASIC-52 does not have the usual LOAD and SAVE instructions. These are replaced after a fashion by the ROM and PROG instructions. The function of the CHAIN instruction is partly reproduced in BASIC-52's RROM instruction. Modified LOAD and SAVE instructions are provided as discussed below.

The following BASIC commands assume a somewhat different form in BASIC-52:

WHILE...WEND, PEEK, POKE, AND, OR, XOR, LPRINT, LLIST

WHILE...WEND: This function is replaced by the "DO" loop instruction which has two forms; DO...WHILE and DO...UNTIL, where the statements to be repeated appear between the DO and the WHILE or UNTIL as follows:

Example:

```
10 DO                    10 DO
20 PRINT A,              20 PRINT A,
30 A=A+1                 30 A=A+1
40 WHILE A<10            40 UNTIL A=10
```

Displayed results for either case:

0 1 2 3 4 5 6 7 8 9

PEEK, POKE: These BASIC functions are replaced in the 8052 by the XBY function.

Logical operators AND, OR, and XOR take the form .AND., .OR., and .XOR. in BASIC-52.

Printer port commands LLIST and LPRINT are replaced by LIST# and PRINT#. Note that the printer port must first be configured with the BAUD instruction.

The following BASIC commands have a somewhat different function in BASIC-52:

INT, LIST

The INT command has the same function as the FIX command found in other BASICs. Fractions are truncated, not rounded.

The 8052 LIST command takes only three forms:

LIST	The entire current program is listed
LIST [x]	The program is listed from line number "x"
LIST [x]-[y]	Line numbers "x" through "y" are listed

References:

EDIT, RENUM	Section 3.3: Editing Programs
ROM, PROG:	BASIC-52 USERS MANUAL - Chapter 3
LOAD, SAVE:	Section 3.2: Memory Management
RROM:	Section 3.2.2: Recalling programs

3.1.4 HARDWARE DEVICES

BASIC-52 has no capacity to handle "files", notably I/O files. The following common file control commands do not appear in BASIC-52.

FILES, OPEN, CLOSE, INPUT#, INP, OUT, EOF, LOF, LOC, WRITE, GET, PUT

Programming to control hardware devices is implemented by custom B52 instructions, CALLs, or writing control and data information byte by byte to registers in these devices.

Device registers are located in the 64K byte memory space of the Model 52 starting at address F000 hex. They include the auxiliary RS232 ports, counter/timers, various I/O ports, and analog devices.

As for the keyboard, single character keystrokes may be captured as ASCII code numbers with the GET function. This is similar to the INKEY$ function of other BASICs.

References:

Hardware devices:	Section 4.1: I/O Devices
	Appendix B: Memory Map
	Section 4.2: Custom Instructions
CALLs:	Section 4.4: Machine code

3.2 Program and Memory Manipulation

3.2.1 GENERATING AND EDITING PROGRAMS

There are two ways to get a program into the B52. New programs are typed in from scratch, or downloaded from another computer. Small programs are usually typed in while in the Direct command mode. Larger programs are best created on another computer where the programmer can take advantage of sophisticated editing software and "keyboard macros" to avoid the repetitive typing of often used B52 commands.

Otherwise, the programmer must suffer with the limited editing capability of BASIC-52.

There is no way to edit a line once Return is entered.
The only recourse is reentering that line.
There is no renumbering facility common to other BASICs.

Tools that are provided include Backspace (ASCII 127) which appears as "Rubout" on some terminals. Also, a "delete line" facility is provided. A "Control D" will erase the line currently being typed if Return has not yet been entered.

Model 52 programs are ASCII files which can be manipulated by most editing software, even other Advanced BASIC software such as Microsoft BASIC.

Advanced BASIC can be used to renumber (RENUM) and MERGE B52 programs if the programmer is careful about restoring any code altered in this process. Microsoft BASIC has trouble with the BASIC-52 ".OR." and ".AND." functions, large numbers, and hex numbers. Look for the telltale exclamation marks "!". Code alterations are easily fixed and renumbered programs then can be SAVEd using the ASCII save option.

Example: SAVE "B52.PRG",A

References:

Memory: Appendix B: Memory Map

3.2.2 SAVING PROGRAMS

To save a program in EEPROM, you ordinarily use one of the "PROG" commands. In response, the B52 stores the program which is currently in RAM memory. This process takes quite a while for longer programs. During the storing operation, the the B52 displays the number assigned to the program. Multiple programs can be saved up to the limit of EEPROM memory space.

EXAMPLE:
Once your first program is entered (and is known to work), in the Direct command mode;

```
You type:          >PROG <cr>
B52's Response:      1

at length...       READY
                   >
```

At this point, the B52 has allocated the necessary amount of non-volatile memory space, stored the program, and labeled it program number 1. The next program stored in EEPROM will be labeled number 2.

The easy way to deal with EEPROM program control is to use the DIRECTORY program to find out what's in there, and use the SAVE instruction provided to augment the BASIC PROG instructions. While the PROG instruction merely adds the program currently in RAM to the end of the list of stored EEPROM programs, SAVE allows you to select which location will be used for storage. This instruction avoids the storage of multiple copies of a single slightly modified program.

Program boundaries are marked with an end of program character (01 hex) followed by a begin program character (55 hex), if any.

The factory supplied DIRECTORY program allows examining the programs in EEPROM, and running programs stored there. In the Direct mode, type LOAD to run the DIRECTORY program.

This DIRECTORY program will display the first line of any stored program that begins with a "REM" statement, and so the first line should be reserved for the program title.

References:

PROG:	BASIC-52 USERS MANUAL: Chapter 3.3
SAVE:	Section 4.2.12
Accessing Memory:	Section 3.2.5

3.2.3 RECALLING PROGRAMS

To access a program from EEPROM, the ROM command is used. This will access the first stored program. For access to other stored programs, the ROM command is followed by a program identification number. The command "ROM 3" will access the third program in EEPROM. Following this instruction, the program can be LISTed.

To edit a stored program, it must be transferred to RAM with the XFER command. The XFER command places the selected program into RAM, automatically switching to RAM mode. It is not necessary to transfer the program selected with the ROM command into RAM to run the program. The selected program may be run by issuing the RUN statement immediately following the ROM command. This allows you to have a program residing in RAM while accessing and running a different program directly from EEPROM. The program residing in RAM will remain intact while a program is running in EEPROM.

> Note: It is not possible to edit, or download an external program while in ROM mode. If downloading is attempted, no error message will be generated. Any subsequent RUN instruction will run the wrong program. The download is lost.

When running a program from EEPROM, it is possible to exit and run a different EEPROM program without stopping. The RROM command does this. The programmer must specify the identification number of the target program. No program variables can be passed from one program to another.

References:

ROM:	BASIC-52 USERS MANUAL: Chapter 3.1
XFER:	BASIC-52 USERS MANUAL: Chapter 3.2
RROM:	BASIC-52 USERS MANUAL: Chapter 4.35

3.2.4 AUTOMATIC OPERATIONS

Once the Model 52 is programmed, it is possible to automatically recall and run a program on power up. This can eliminate the need to have a remote console for operator input. It is neccessary to have the program you wish to automatically recall and execute at the first EEPROM location (ROM 1). It is therefore important to make sure that the first program stored with the PROG command is the program that you wish to automatically recall and execute.

To enable this "autorecall autorun" function, enter the PROG2 command. After the PROG2 command is entered the B52 will automatically recall and run the first stored program every time it is powered on. The PROG2 command also saves the current console communication protocol. Thus, console communications are automatically initialized on power up. In this mode, a console may not even be needed if no operator input is called for.

The PROG1 command option allows automatic console initialization without running a program. The PROG1 command stores the current baud rate information in EEPROM and automatically initializes the console on power on. This eliminates the need to send a space character before beginning operation.

BASIC-52 provides other PROG options, up to PROG6, but the value of these functions is obviated by the machine language power-up routines executed by the B52. Do not use them.

References:

ROM:	BASIC-52 USERS MANUAL: Chapter 3.1
PROG:	BASIC-52 USERS MANUAL: Chapter 3.3

3.2.5 ACCESSING MEMORY

The 8052 processor has the capacity to access a full 128k of memory, half "Data Memory" (RAM: 0 to 64k), half "Program Memory" (ROM: 0 to 64k). The Model 52 architecture has RAM space from address 0 up to 32k and ROM also from 0 to 32k. The first 8k of ROM is internal to the 8052 (BASIC-52), the remaining 24k is permanent EPROM.

EEPROM (and the I/O devices) occupy the space from 32k up. EEPROM can be treated as either RAM or ROM. The B52 has 16k of both RAM and EEPROM installed.

> NOTE: The processor itself also has a separate 256 bytes of internal RAM. This memory is almost exclusively used by the BASIC-52 interpreter. Ordinarily the programmer will not need to access this memory. BASIC-52 provides the "DBY" instruction to do this.

BASIC-52 provides the "XBY" instruction to access RAM.

Example: `PRINT XBY(1000H)`

(display the contents of Data Memory location 1000 hex)

`XBY(8000H)=123`

(set memory location address 1000 hex to the value 123)

This instruction can also be used to access the I/O devices if desired. This is ordinarily unnecessary when the I/O instructions described in Section 4.2 are used.

> CAUTION: The B52 uses 612 bytes of memory at the top of RAM (3D9Bh to 3FFFh) for storing operating parameters. Do not write to this space. Program variables are stored at the top of memory and just above the program. BASIC stores two addresses representing the boundaries of variable storage space. See BASIC-52 USERS MANUAL: Appendix 1.7

BASIC-52 provides the "CBY" instruction to access ROM.

Example: PRINT CBY(4000H)

(display the contents of Program Memory location 4000 hex)

B52 circuitry is such that both the 8052 PROG and XBY instructions will write to EEPROM and both XBY and CBY operators will read from it. Only XBY works in RAM, only CBY works in EPROM.

> NOTE: When writing more than one byte to successive memory locations in EEPROM, it is necessary to allow sufficient time for data to be stored. To guarantee that data is properly stored, the program should wait and test for verification after each write before writing again.
>
> Example:

```
100 XBY(8500H)=X1                      (write to EEPROM)
110 IF XBY(8500H)<>X1 THEN 110          (wait for verification)
120 XBY(8501H)=X2                      (write the next byte)
130 (wait again)
```

BASIC-52 allows the operator to run programs in EEPROM without first loading them into RAM. This is the result of BASIC-52's two operating modes which are called RAM mode and ROM mode. The instructions RAM and ROM allow transferring between modes. No editing or downloading is possible in ROM mode.

Normally a PROG instruction is used to save a BASIC-52 program, but repeated PROG instructions during editing will result in multiple saved copies of the same program. The B52 has a SAVE instruction whereby the programmer can specify where in EEPROM the program should be saved. Note that execution of a SAVE may involve substantial relocation of currently stored programs, resulting in long delays. To avoid this, use the LOAD function and DIRECTORY program to keep track of the contents of programs from EEPROM.

> NOTE: The LOAD instruction will erase any program currently in RAM. You cannot LOAD to inspect the DIRECTORY and then SAVE or RUN the program you had in RAM.

The DIRECTORY program displays the start address of all stored programs. A program and all those above it can be erased by setting the start address location to zero:

XBY(8010H)=0 (erase all stored programs)

Most dedicated control applications will have B52 programs running automatically from EEPROM, while the RAM is used only for data storage and manipulation.

B52 permanent EPROM contains a number of handy machine language routines to enhance I/O control, as well as the custom instructions that have been added to BASIC-52.

The B52 repertory of demonstration programs also resides here. These may be accessed in the Direct command mode by issuing the LOAD instruction. The factory supplied programs below are provided. Program listings appear in Part IV.

DIRECTORY - This program shows what is saved in EEPROM
TEST - This program provides I/O test and demonstration
INDEXER - This program provides test and control of Compumotor models CX, 372, and 2100

References:

RAM, ROM:	BASIC-52 USERS MANUAL: Chapter 3.1
LOAD, SAVE:	Section 4.2: Custom Instructions
XBY, CBY:	BASIC-52 USERS MANUAL: Chapter 7.1
Memory:	Appendix B: The Memory Map

3.3 Input/Output Programming Fundamentals

The following discussion deals with the binary nature of the microcomputer. The the operations and functions described are implemented by the custom instructions and CALL functions of the B52 discussed in Section 4.2.

All I/O functions in the Model 52, from turning on an output to receiving a transmitted character require that the processor handle a "byte" of data. This is an 8 bit quantity, reflecting the 8 bit data bus of the processor. "Bit" means binary digit.

When the processor "reads" the 8 data bus inputs from an I/O device, the state of those inputs may reflect the status of 8 external switches, the current value of an event counter, or an RS-232 character code. The processor treats such a byte of data as a binary number between 0 and 255. In BASIC these bytes can be handled as decimal or hexadecimal numbers; decimal for floating point math and hexadecimal for logical control of discrete I/O.

3.3.1 BASIC I/O READ AND WRITE INSTRUCTIONS

The hardware I/O devices in the Model 52 all have internal registers with "addresses" in its memory space. The 8052 BASIC "XBY" instruction is used to read from or write to any address whether a hardware device or a memory location is at that address.

For example, the Analog to Digital input device is located at address F200 hex. The Digital to Analog Converter output device is at address F400 hex. To set the analog output, a number corresponding to the desired output voltage (say 128 for 0 volts out) is written to the designated address as follows:

```
XBY( 0F400H ) = 127
```

(write the value 127 to address F400 hex)

To read an analog input, a number is read from the designated address:

```
A = XBY( 0F200H )
```
(set variable A to the value found at address F400 hex)

Ordinarily, the "ANLG" instruction would be used for analog operations.

References:

Addresses:	Appendix B: Memory Map
XBY, CBY:	BASIC-52 USERS MANUAL: Chapter 7.1
ANLG:	Section 4.2.1

3.3.2 BITS, BYTES, AND DISCRETE I/O LOGIC

Each discrete input and output in the Model 52 is part of a group or channel of eight. These I/O channels will be referred to as "ports". I/O Ports are read from or written to a byte at a time. To examine a single input, or switch a single output, it is necessary to separate it from the other seven. This is accomplished by logically "masking" the desired bits.

This can result in some laborious programming that runs comparatively slowly. The B52 has two custom instructions that eliminate the need for most bit masking operations when handling discrete I/O. These are the DIO and SCAN instruc-

tions. These instructions automatically handle the "negative true" nature of B52 discrete I/O discussed below. The discussion on boolean logic that follows is for information.

B52 discrete I/O assumes Negative True logic on all discrete I/O operations except the *ISOLATED* outputs. Standard signal conditioning equipment requires a low voltage output to sink current and drive the output module. To turn on a single output, it is necessary to set the appropriate bit of the output port register to 0. The corresponding *PARALLEL OUTPUT* voltage will go from high to low (the *ISOLATED* outputs, however, require a 1 to turn On and conduct current).

By the same token, inputs are normally high (1) if unconnected, and go low (0) when input devices turn On. This is true for all discrete inputs.

Two special CALL instructions are provided to invert input and output operations for the DIO and SCAN instructions when Negative True logic is not desired. See Section 4.4.

Any byte may be logically inverted using the ".XOR." function discussed below.

Register bits are numbered 0 through 7, where bit 7 is the most significant bit (MSB) . To set bit number 5 to a 1 while the other seven remain 0 is a simple matter of writing the binary number

(MSB) 0 0 1 0 0 0 0 0 (LSB)

to the register. This binary number has a decimal value 32, hexadecimal 20. This value is simply 2 raised to the power 5, 5 being the number of that bit.

Example:

bit 3 has the value 8, 2 to the third power (2 cubed)
bit 7 has the value 128, 2 to the 7th power
the maximum value of any byte is 255, with all bits set:

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1 = 255

To handle binary numbers, the programmer uses hexadecimal notation which makes it much easier to convert from the binary numbers, or visualize discrete I/O.

A program statement would have the form "output port = 37 hex". Any pattern of outputs can be set by writing a number to the port in this way.

In practice, it will be necessary to turn an output on or off without changing the others in the same port. To do so, the programmer must know the current status of the port, in order to logically determine the new status once the bit is changed.

3.3.3 SETTING BITS: THE LOGIC "OR" FUNCTION

To set a register bit, determine the hex value of a byte where only the desired bit is set as in the case above. This number becomes the logical mask.

To set a bit in a byte register (or port output) to a 1, read the target byte value, logically OR the mask with the byte value, and write the result back to the register (or port).

Example:

The Model 52 has a parallel output port at address F600 hex. The following statement would turn off output 5 (set the port register bit 5 to a 1), without affecting the other outputs of the port:

```
XBY( 0F600H ) = 20H .OR. XBY( 0F600H )      (set bit 5)
```

3.3.4 CLEARING BITS: THE LOGIC "AND" FUNCTION

To clear (reset) a register bit and set it to a 0, the logical mask must be set to the inverse of the mask for the OR operation. In the above case, this would be

```
(MSB)    1 1 0 1  1 1 1 1    (LSB)
```

which is equivalent to hexadecimal DF. To reset the register bit (and turn on output 5), read the current value, logically AND this mask with the value, and write the result back to the register. Syntax for the output port example is:

```
XBY( 0F600H ) = 0DFH .AND. XBY( 0F600H )      (reset bit 5)
```

3.3.5 INVERTING: THE "XOR" LOGIC FUNCTION

Outputs may be simply inverted, or "toggled" using the XOR (Exclusive OR) function. The mask is the same as for the OR function above. This logical operator may also be used to invert an entire byte, useful for converting from negative to positive true logic. This requires performing an exclusive OR of the byte with FF hex (all ones). Syntax for printing an inversion of the byte read from the above port:

```
PRINT XBY( 0F600H ) .XOR. 0FFH
```

3.3.6 READING INPUTS

Inputs are read from 8 bit ports also. A single input may be examined by masking off the other seven bits (reset them to 0). This requires a single bit mask like that used above for the OR function, with the AND function. To read any port input, read the port register, AND the mask with the port value, and see if the result is zero. If so, the input is On.

The Model 52 has an parallel input port at address F800 hex. The following statement would display the state of input 5:

```
IF (XBY(0F800H) .AND. 20H) = 0 THEN PRINT "OFF!"
      ELSE PRINT "ON!"
```

3.3.7 INPUT DECISIONS

In practice, using the above technique to test a large number of interdependent inputs can result in a large and difficult program. The SCAN instruction can greatly simplify this kind of requirement. Otherwise, when more than one input is involved in the decision about what to do with the outputs, it becomes practical to handle the input port byte values as numbers.

In any program, the state of an input port is assigned to a program variable. The BASIC ON (variable) GOTO ... or ON (variable) GOSUB ... statements can be used to good advantage to organize program code in a decision process based on an input port. This helps insure that all possible combinations of input states are properly dealt with.

References:

I/O Port Addresses:	Appendix B: Memory Map
DIO, SCAN:	Section 4.2:Custom Instructions
.OR.,.AND.,.XOR.:	BASIC-52 USERS MANUAL:Chapter 5.1