Publication Number 420305398-001A
Order Number 420305398-001
August 1977

Simple Cost-effective Microprocessor (SC/MP)

SC/MP NIBL
REFERENCE GUIDE

2900 Semiconductor Drive
Santa Clara, California 95051

PREFACE

This reference guide for the National Industrial Basic Language Interpreter (NIBL, pronounced "nibble") is a version of TINY BASIC; refer to Doctor Dobbs Journal, Volume 1, January 1976. NIBL is intended for moderate-speed input/output and control applications utilizing either a SC/MP or a SC/MP-II microprocessor.

Included in the NIBL Reference Guide is information pertaining to NIBL program-entry, program control, and program content. The appendices provide information on NIBL Formal Grammar, NIBL Error Messages and Descriptions, and "BAGELS" — A simple NIBL game.

Versions of NIBL are available for both SC/MP (2-microsecond microcycle) and SC/MP-II (1-microsecond microcycle). Versions for these two processors differ only in delay constants contained within their Teletype® driver routines. These Teletype drivers contain software timing loops. For SC/MP-II, two standard products are available:

- ISP-8F/351 Eight 512 x 8 (MM5214) ROMs (see figure 1-1)
- ISP-8F/352 Two 2K x 8 (MM2316A) ROMs (see figure 1-2)

For either SC/MP or SC/MP-II, paper tape versions are available through COMPUTE, the National Microprocessor Users Group Newsletter. Following are the order numbers for these versions:

- SL043A-P for SC/MP
- SL043A-N for SC/MP-II

The material within this publication is subject to change without notice. Changes will be reported in COMPUTE.

Copies of this publication and other National Semiconductor publications may be obtained from the National Semiconductor sales office or the distributor serving your locality.

Related Publications:

- SC/MP Data Sheet, ISP-8A/500D Single-Chip 8-bit Microprocessor — Publication Number 420305227-001 — No charge. *
- SC/MP-II Data Sheet, ISP-8A/600 Single-Chip 8-bit (N-Channel) Microprocessor — Publication Number 426305290-001 — No charge. *
- SC/MP Technical Description — Publication Number 4200079 — Cost $3. *
- SC/MP Assembly Language Programming Manual — Order Number ISP-8S/994Y — Cost $10, order from local distributor.
- SC/MP Kit Users Manual — Publication Number 4200113 — Not available separately; supplied in SC/MP Kit.
- Dr. Dobb's Journal of Computer Calisthenics and Orthodontia — Volume 1, Number 1, January 1976 and Number 10, November 1976.

  PCC
  Box Number E
  Menlo Park, CA 94025

- SC/MP Microprocessor Applications Handbook — Publication Number 420305239-001A — No charge. *
- SC/MP Low Cost Development System (LCDS) Users Manual — Publication Number 4200105A — Cost $10, order from local distributor.

* Contact: Marketing Services
National Semiconductor Corporation
2900 Semiconductor Drive
Santa Clara, California 95051

or

Telephone: (408) 737-5142

® Registered trademark of the Teletype Corporation

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Chapter 1

GENERAL INFORMATION

1.1 INTRODUCTION

The National Industrial Basic Language (NIBL) provides the user with a version of the TINY BASIC Programming Language intended for moderate-speed input/output and control applications utilizing either a SC/MP or a SC/MP-II.

1.2 SYSTEM DESCRIPTION

Because NIBL initializes in a logically clear state, the first requirement is to supply a program; refer to chapter 2 for a detailed discussion of PROGRAM ENTRY. Chapter 2 also contains descriptions of various initialization commands and their uses.

Chapter 3 provides explanations of NIBL variable names, relational operators, expressions, standard arithmetic and logical operators, and decimal and hexadecimal constants, and a discussion of several functions which may be included in NIBL arithmetic expressions. Chapter 4 includes descriptions of input/output and assignment and control statements. Chapter 5 provides a discussion of the indirect operator.

The last two chapters (chapters 6 and 7) include discussions of multiple statements on a single line, multiple-page handling, string handling, and explanations of the LINK, REMARK and END Statements.

The NIBL formal grammar, NIBL error messages and descriptions, and "BAGELS" — a simple NIBL game — are included in appendices A, B, and C, respectively.

1.3 HARDWARE CONFIGURATION

1.3.1 Minimum Hardware Configuration

The minimum hardware configuration required to support the NIBL Interpreter is as follows; also refer to figures 1-1 and 1-2.

a. SC/MP Central Processing Unit (CPU), crystal, and support logic.

b. 110-Baud ASCII terminal interface.

c. 4K-by-8 Read-Only Memory (ROM) (0-FFF) for NIBL memory.

d. 2K-by-8 Read-Write Memory (RAM) (1000-17FF) for user memory (allows approximately 60 average NIBL line statements).

e. Model 33 ASR Teletype (TTY) or similar terminal.

f. Power supplies (+5, -12 volts).

Users who do not wish to build a system from scratch can obtain items a through d above already assembled and tested in the form of the SC/MP Low Cost Development System (ISP-8P/301), the 4K-by-8 ROM/PROM Card (ISP-8C/004P), and the 2K-by-8 RAM Card (ISP-8C/002). The blank MM5204Q PROMs (included in the ISP-8C/004P) then may be programmed with NIBL using the paper tape that is available from COMPUTE, the Microprocessor Users Group newsletter.

NOTE

The following SC/MP Status Register bits for Teletype interfaces are not available to the user's NIBL program: User Flag 0, User Flag 1, and Sense Bit B.

1.3.2 Additional Memory

Another 2K bytes of RAM (1800-1FFF) will expand the storage capacity to accommodate programs of approximately 160 lines. The total 4K bytes of RAM hereafter is referred to as PAGE 1. NIBL allows the user to add on six more 4K bytes of pages, providing a total of 28K bytes of user memory that can be used to store NIBL programs. Each 4K bytes of user memory can contain a separate NIBL program, and control can be transferred from one page to another during execution.

1.3.3 Using a SC/MP-II

If a SC/MP-II is used, a 4.0-megahertz crystal is required because the SC/MP-II increased speed-of-operation can be up to a maximum of twice that of the SC/MP speed-of-operation. The delay parameters in the Teletype Input/Output Routines differ between versions of NIBL operating on SC/MP and SC/MP-II. The COMPUTE Users Library has two versions of NIBL, for SC/MP and SC/MP-II: the version released and supported by National Semiconductor will run only on SC/MP-II.

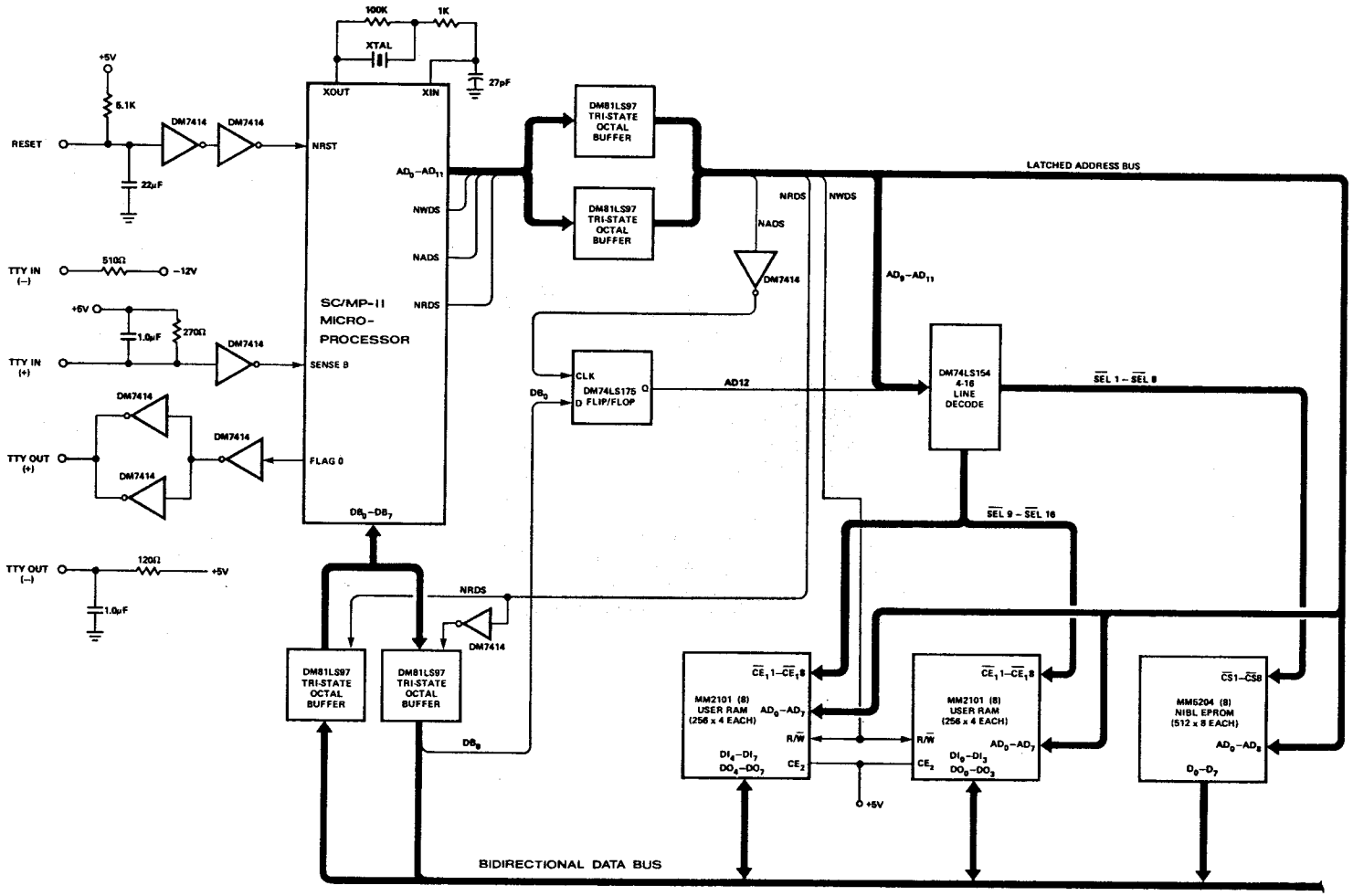

Figure 1-1. Minimum Hardware Configuration to Support the NIBL Interpreter (ISP-8F/351)

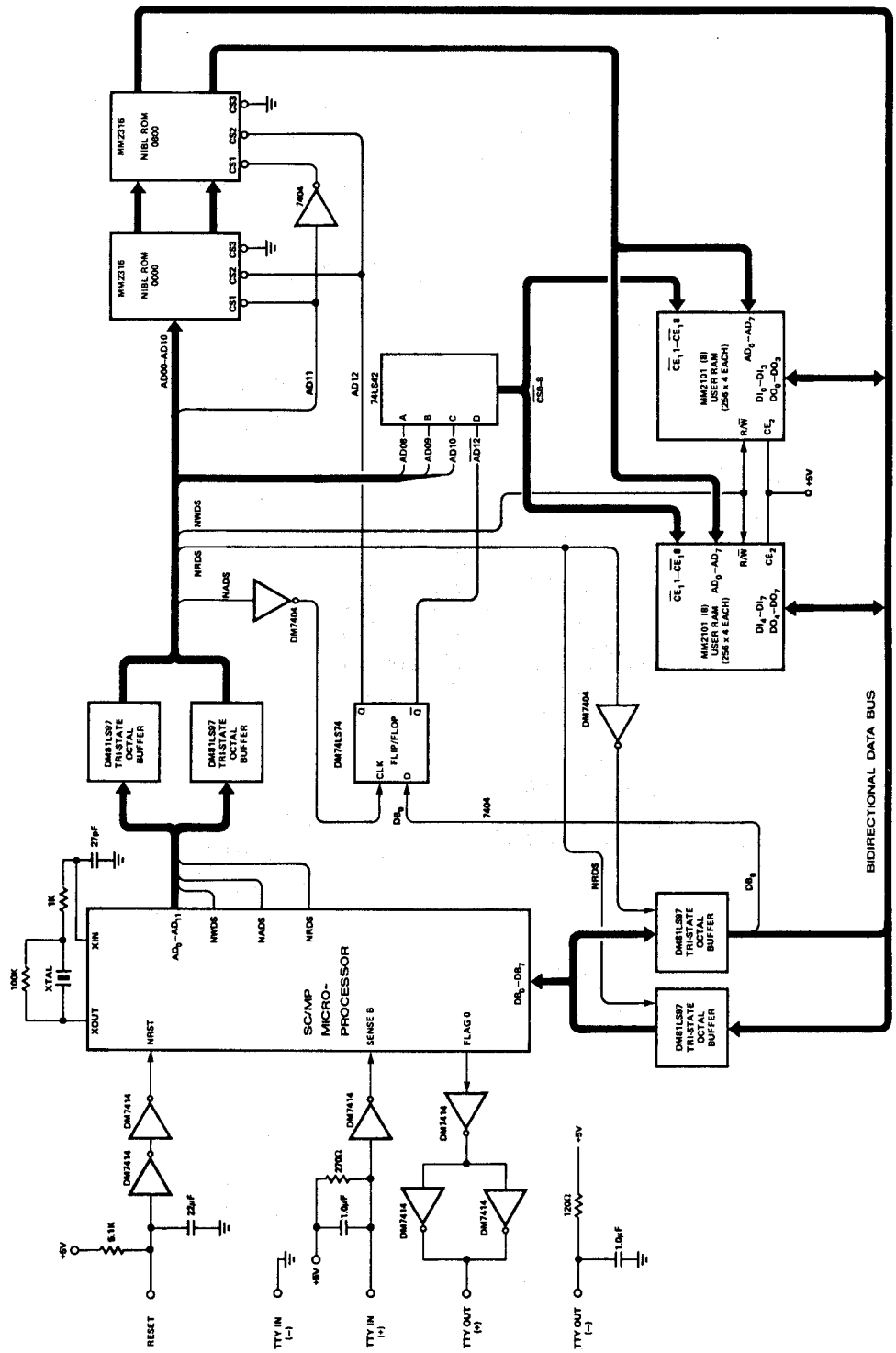

Figure 1-2. Minimum Hardware Configuration to Support the NIBL Interpreter (ISP-8F/352)

Chapter 2

PROGRAM ENTRY AND CONTROL

2.1 PROGRAM ENTRY

NIBL initializes in a logically clear state so that the first requirement is to supply a program. This is accomplished either by typing in numbered lines, or by reading in a previously prepared program through the Paper Tape Reader. NIBL doesn't recognize the difference between these two methods — it will read characters from the paper tape if the Reader is ON; otherwise, the only input will be through the Keyboard. As is customary with Teletypes, there is no distinction between tape and keyboard input, so typing while the Reader is running will produce errors.

2.1.1 Prior to Program Entry

Before entering a new program, type the command NEW . This procedure ensures that NIBL is ready to accept a new program.

2.1.2 Entering Program Lines

Lines may be entered in whatever order is convenient; the actual order in the program is determined by the line number, which must begin each line that is to be placed in the program — and not by the order in which the lines are received. Leave an interval of at least 10 lines between line numbers because it may be necessary to insert lines later; there is no way to renumber lines except to retype them. Leaving gaps in the line numbering is reasonable since there are 32K possible line numbers.

2.1.3 Punctuation with Blanks

Punctuation with blanks is according to taste, as long as keywords are not broken up. Thus, though few blanks are required, and all are stored with the program, blanks to provide clarity should be included.

2.1.4 Additional Proper Procedures

A line without a number executes immediately, but entering a number without a line will delete that line, if it exists. This is permissible since it is the only way to dispose of a specific line; however, if the user starts to type a line number and then changes his mind, he must be sure to hit a CONTROL/U , NOT the Carriage Return (CR) Key, as it may cause a good line to disappear or be replaced by a mangled line. NIBL doesn't check the correctness of lines as they are entered, so a mangled line will act as a hidden mine which will blow up the program when it attempts to execute.

2.1.5 Control Characters

There are certain control characters which cannot appear in a NIBL text. These are described as follows:

CONTROL/U.	Typing this character causes the current line to be deleted.
CONTROL/C.	Typing this character causes NIBL to resume accepting program lines or immediate commands. Use CONTROL/C to abort program execution from an input statement response.

SHIFT/O (Back-Arrow).	This character deletes the last character typed in (except for another back-arrow), and may be used repetitively.
CONTROL/H (Back-Space).	This character is identical in function to SHIFT/O, and is intended for use with video terminals. It echoes as back-space/space/back-space.
Carriage Return.	This character terminates the current line and enters it to the NIBL system.

Line Feed and Null Characters are echoed at the Teletype by NIBL, but are ignored. This allows the user to save a program by listing it with the Paper Tape Punch turned ON; the tape can be entered to NIBL at any time through the Paper Tape Reader.

Program execution can be halted by pressing any key; however, NIBL checks for this only between each statement, so to halt program execution, the BREAK key on the Teletype is pressed. Pressing any key while NIBL is listing a user's program will cause the listing to be terminated.

2.1.6 Program Entry Summary

- A line without a number executes immediately.
- A line with a number is inserted in order in the program.
- Line numbers must be from 0 to 32767.
- No blanks in keywords (LET, IF, THEN, GOTO, GOSUB, GO, TO, SUB, RETURN, INPUT, PRINT, LIST, CLEAR, RUN, and so on).
- Blanks outside keywords are optional.
- SHIFT/O (Back-arrow) deletes the last character typed.
- CONTROL/H (Backspace) has the same function as SHIFT/O (for use with CRT's).
- CONTROL/U deletes the entire line.

2.2 PROGRAM CONTROL

2.2.1 NIBL Commands

2.2.1.1 NEW Command

The command NEW should be typed on the TTY prior to entering a program, but the user must be reminded that any existing program is lost after entering (CR), the Carriage Return Key. Fortunately, the NEW Command is not legal as a line in a NIBL program.

Typing NEW followed by a Page Number clears the program in that PAGE.

2.2.1.2 RUN and GOTO Commands

Using the RUN Command is not the only way to start a program; that is, if 10 is the lowest line number, GOTO 10 will accomplish the same thing. If it is desired to start at some line, say 110, not the first line in the program, just type GOTO 110 instead of RUN.

The RUN Command clears all internal stacks and variables, the GOTO Command does not.

2.2.1.3 LIST Command

Typing [LIST] causes the NIBL program in the current PAGE to be listed in its entirety. If the user wants to check around line 110, type [LIST 90] and hit the BREAK key when enough of the program has been listed. Beyond its primary use, to display the program, LIST serves, on an ASR-33 Teletype, as the method for saving a program: type [LIST] ; then turn on the Punch; then hit (CR), the Carriage Return Key.

2.2.1.4 CLEAR Command

The CLEAR Command will cause all variables (A-Z) to be zeroed. CLEAR also clears all the stacks used by a NIBL program indirectly; that is, the GOSUB stack, the DO/UNTIL stack, and the FOR/NEXT stack.

2.2.2 Program Control Summary

- CLEAR clears all variables (A-Z) and all stacks.
- NEW deletes the program in PAGE 1 (normal page).
- NEW n (n is from 2-7) deletes the program in PAGE n.
- LIST lists the program starting at the lowest line number, or the line number given.
- RUN starts the program at the lowest line number.

NOTE

Repeated pressing of the BREAK Key interrupts execution and returns the system to programming mode.

Chapter 3

NIBL EXPRESSIONS AND FUNCTIONS

3.1 EXPRESSIONS

3.1.1 Variable Names

In NIBL there are 26 variable names, that is, the single letters of the alphabet. The variables are all 16-bit signed INTEGERS; there are no fractions or floating point numbers in NIBL. All numeric constants are decimal numbers, except when preceded by a pound sign, '#', in which case the constant is taken to be hexadecimal.

3.1.2 Relational Operators

The Relational Operators ("=", "<", ">", "<=", ">=", and "< >") are the standard BASIC types and should be self-explanatory, except for '< >', which means "is not equal to". Note that '> <' is illegal. The Relational Operators return either 0 (FALSE) or 1 (TRUE) as a result.

3.1.3 Arithmetic Operators

The standard operators '+', '-', '/', and '*' are provided.

Arithmetic is standard 16-bit twos-complement arithmetic. Fractional quotients are truncated, not rounded; division by zero induces an error break. The order of evaluation of arithmetic expressions in NIBL is controlled by parentheses; operator precedence also influences operations, but is is safer to use many parentheses in complicated expressions to avoid confusion.

3.1.4 Logical Operators

In addition to the standard arithmetic operators, NIBL provides Logical Operators AND, OR, and NOT. The first two are binary operators like * and +; the last is a unary operator like @. These operators perform bitwise logical operations on 16-bit arguments, producing 16-bit results.

3.1.5 Expression Summary

- All expressions are 16-bit, twos-complement values.
- 26 variable names: A through Z.
- Relational Operators <, >, =, <=, >=, < >.
- Arithmetic Operators +, -, *, /.
- Logical Operators AND, OR, NOT.
- Decimal Constants in the range -32767 to 32767.
- Hexadecimal Constants denoted by '#' followed by hex digits.

3.2 FUNCTIONS

There are several functions which may be included in arithmetic expressions in NIBL. These are described as follows.

3.2.1 MOD Function

MOD (a,b) returns the absolute value of the remainder of a/b, where a and b are arbitrary expressions. If b is zero, (0), an error break will occur.

3.2.2 RND Function

RND (a,b) returns a pseudo-random integer in the range of a through b, inclusive. For the function to work correctly, b-a must be less than or equal to 32767 (base 10), and b should be greater than a.

3.2.3 STAT Function

STAT returns the value of the SC/MP Status Register (an 8-bit value). Note that STAT may appear on the left side of an Assignment Statement as well as on the right side. The Carry and Overflow Flags of the status register are usually meaningless, since the NIBL Interpreter itself is continually modifying these flags. The Interrupt-Enable Flag cannot be altered by an assignment to STAT (as an example: STAT = #FF); NIBL clears that bit when making any assignment to the status register. User Flags 0 and 1 and Sense Bit B are not available to the user's NIBL program for Teletype interfaces.

3.2.4 TOP Function

TOP returns the address of the first byte above the NIBL program in the current Page which is available for use. In other words, it is the address of the highest byte in the NIBL program, plus 1. All the memory in the current Page, above and including TOP, can be used by the NIBL program as scratch storage.

3.2.5 PAGE Function

PAGE is a pseudo-variable (like STAT) which refers to the 4K memory page in which a NIBL program is either being edited or run. The only legal values for PAGE are 1 through 7; when a number is assigned to PAGE which is outside these limits, only the least significant three bits of the number are used. If the resulting value is 0, the value 1 is assumed. As an example, PAGE = 3000 would cause PAGE to be set to 1; refer to 6.2, Multiple-Page Handling, for more information.

3.2.6 Function Summary

- RND (a,b) returns the random number in the range a through b.
- MOD (a,b) returns the remainder of a/b.
- STAT returns the value of the SC/MP Status Register.
- PAGE returns the number of the current Page.
- TOP returns the highest address of NIBL program in the current Page.

3.3 HIERARCHY OF OPERATIONS

The order of performing individual operations within an expression is determined by the hierarchy of operators and the use of parentheses. Operations of the same precedence are performed from left to right in an expression. Operations within parentheses are performed before operations not in parentheses. The hierarchy (from highest to lowest) of operators is as follows:

NOT, functions, @, ( )

*, /, AND

+, -, OR

<, >, <=, >=, < >

For example, the expression 2 + 3 * 4 would have the value 14, because the '*' operator has a higher precedence than '+'. Thus, the term 3 * 4 is computed first, giving a result of 12; then the expression 2 + 12 is computed, giving the final result of 14. On the other hand, the expression (2 + 3) * 4 would have the value 20, since NIBL computes the value of the subexpression in parentheses before evaluating the surrounding expression.

Chapter 4

INPUT/OUTPUT, ASSIGNMENT, AND CONTROL STATEMENTS

4.1 INPUT/OUTPUT STATEMENTS

4.1.1 Executing an Input/Output Statement

Input of all numbers to NIBL may be accomplished by executing a statement such as

```
INPUT A,B
```

When this statement is executed, NIBL prompts the user with a question mark. Now the user types in two expressions, separated by commas, which will be assigned to the variables A and B. Thus, a legal response to the INPUT statement above would be

```
45,#FFC0
```

String input is also allowed in NIBL; refer to 6.3, String Handling, for more information.

Typing Control/C during an INPUT response causes NIBL to abort execution of the program and to return to the Edit Mode.

4.1.2 Output Statements

In Output Statements, quoted strings (such as "THIS IS A STRING") are displayed exactly as they appear (with the quotes removed). Numbers are printed in decimal format, with either a leading space (for positive numbers) or a minus sign, '-', for negative numbers; and a trailing space for all numbers. A semicolon (;) at the end of an Output Statement suppresses the usual Carriage Return with which NIBL terminates the output.

Strings in memory (such as those generated by a String Input Statement) may also be printed; refer to 6.3, String Handling, for more information.

4.1.3 Input/Output Statement Summary

- INPUT X
- INPUT X,Y,Z
- PRINT "A STRING"
- PRINT "F =",M*A
- PRINT "TAKE",X,"PILLS BEFORE";

NOTE

The semicolon (;) suppresses an otherwise automatic Carriage Return after any Output Statement.

4.2 ASSIGNMENT STATEMENTS

4.2.1 Assignment Statement Forms

The 'LET' in an Assignment Statement may be omitted, that is, the execution of the statement is actually faster if 'LET' is omitted.

The left portion of an Assignment Statement may be a simple variable (A-Z), STAT, PAGE, or a memory location, indicated by a @ followed by a variable, number, or an expression in parentheses; refer to chapter 5, Indirect Operator, for more information.

A String Assignment Statement is also allowed by NIBL; refer to 6.3, String Handling, for more information.

In connection with Assignment Statements, a property may be noted by means of which conditional assignments may be made without using any IF Statements. It hinges on the fact that all predicates, as an example: expressions such as A > B or (A+3)*5 > 100, are actually evaluated to yield a 1 if true, and a 0 if false. Thus, if a predicate is enclosed in parentheses, it may be used as a multiplier in a statement such as:

```
LET X = A*(A >=0)-A*(A < 0)
```

which assigns to X the absolute value of A.

4.2.2 Assignment Statement Summary

- LET X = 7
- E = I * R
- STAT = #70
- PAGE = PAGE + 1
- LET @A = 255
- @(T+36) = #FF

NOTE

'LET' is optional in any Assignment Statement.

4.3 CONTROL STATEMENTS

4.3.1 Control Statement Construction Allowances

NIBL, a superset of the originally-proposed TINY BASIC, allows certain constructions in Control Statements that are not allowed in larger standard BASIC's. These attractive aberrations are in two areas:

A. NIBL allows GOTO or GOSUB to use an arbitrary expression where standard BASIC's would allow only a constant statement number; as an example:

```
      GOTO X+5
or    GOSUB A+(X >Y + 2)
```

B. NIBL allows an arbitrary statement to follow an IF/THEN rather than a statement number to be executed if the predicate is true (as is the rule in standard BASIC's).

NIBL allows the omission of the THEN, as is also allowed in some larger BASIC's — this is one omission which enhances clarity by removing "noise" from the program.

4.3.2 Subroutine Advantages

For users without any programming background, a brief treatment of a "subroutine" (which is what GOSUB and RETURN give to a BASIC) is given. If there is a computation of operation which must be carried out in more than one place in the program, the need to repeat the code involved is avoided by setting this code outside of the program; (as an example: beyond the last statement of the main program) with an additional statement, RETURN, appended. When this has been accomplished such that the first statement of this segregated code (now called a subroutine) is, as an example, 1000, the operations it carries out may be invoked by the use of a GOSUB 1000. The encountering of the RETURN Statement in the 1000 subroutine will cause execution to be taken up at the first line following the GOSUB 1000 last executed. Execution of a GOSUB is usually referred to as calling (or invoking) a subroutine. The space saving and consolidation of code which has been alluded to here as the advantage of the subroutine, is only the slightest suggestion of the immense utility and organizational power of the subroutine concept.

4.3.3 DO and UNTIL Statements

NIBL has another control structure which is not available in any standard BASIC, TINY or otherwise. This is the DO/UNTIL construct, with which program loops may be designed with a minimum of GOTO Statements. The overall effect of this feature is to greatly improve the readability, maintainability, and clarity of NIBL programs.

By enclosing zero or more statements between a DO Statement and an UNTIL <condition> Statement (where <condition> is any arbitrary expression), the user causes these statements to be repeated as a group until the <condition> evaluates to a non-zero number. As an example of the use of the DO and UNTIL Statements, a program which prints the prime numbers is listed as follows:

```
 10  PRINT 1: PRINT 2
 20  I=3                          :REM I IS THE NUMBER BEING TESTED
 30  DO
 40     J=I/2:N=1                 :REM J IS THE LIMIT; N IS THE FACTOR
 50     DO                        :REM TRIES TO FIND A DIVISIBLE FACTOR OF I
 60        N=N+2
 70     UNTIL (MOD(I,N)=0) OR (N>J)
 80     IF N>J PRINT I            :REM DID NOT FIND A DIVISIBLE FACTOR
 90     I=I+2
100  UNTIL 0                      :REM REPEATS THE OUTER LOOP FOREVER
```

4.3.4 FOR and NEXT Statements

NIBL also has FOR and NEXT Statements which are identical to the FOR and NEXT Statements in standard BASIC's. As in other BASIC's, the STEP in the FOR Statement is optional, and if it is included, may be either positive or negative. If the STEP is omitted, a STEP of +1 is assumed. A FOR loop is terminated by a NEXT <var> Statement, where <var> is the variable referred to in the FOR Statement which began the loop. NIBL causes an error break if the variable in the NEXT Statement does not match that in the matching FOR Statement.

FOR/NEXT loops may be nested, and NIBL will report an error if the nesting level becomes too deep. Note that a FOR loop will be executed at least once, even if the initial value of the control variable already exceeds its bounds before starting. As an example of the use of the FOR and NEXT Statements, a program which prints a table of random numbers is listed.

```
10    REM THIS PROGRAM PRINTS A TABLE OF RANDOM NUMBERS,
20    REM USING A SUBROUTINE TO FORMAT THE OUTPUT IN ZONES.
30    FOR K=1 TO 4
40          FOR J=1 TO 8:  N=RND(-16000,16000):  GOSUB 1030:  NEXT J
50          PRINT ""
60    NEXT K
70    END
80    REM
1000        REM THIS SUBROUTINE PRINTS THE VALUE OF N AND SKIPS
1010        REM TO THE NEXT PRINT ZONE (FIELD WIDTH = 8).  THE
1020        REM THE VALUES OF N AND I ARE DESTROYED.
1030        PRINT N;
1040        I=0
1050        DO
1060           N=N/10:  I=I+1
1070        UNTIL N=0
1080        FOR I=1 TO 6-I
1090           PRINT " ";
1100        NEXT I
1110        RETURN
```

4.3.5 Control Statement Summary

- GO TO 15 or GOTO 15
- GOTO X+5
- GO SUB 100 or GOSUB 100
- RETURN
- IF X+Y >3 THEN GOTO 15
- IF X >3-Y GOTO 15
- IF A=B LET A=B-C
- FOR I = 10 TO 0 STEP -2
- NEXT I
- FOR K = 1 TO 5
- DO: X=X+1: UNTIL (X=10) OR (@X=13)

Chapter 5

THE INDIRECT OPERATOR

5.1 INDIRECT OPERATOR USAGE

The Indirect Operator, a powerful feature, is a NIBL exclusive, at least in the realm of BASIC. It realizes the functions of PEEK and POKE operations in other BASIC's, with a less cumbersome syntax, and is a way to access absolute memory locations though its applications are not limited to that. In microprocessors such as SC/MP, where interfacing is commonly carried out via memory addressing, its utility is especially significant.

Preceding a constant, a variable, or an expression in parentheses by an "at" sign ('@'), causes that constant, variable, or expression to be used as an unsigned 16-bit address at which the value is to be obtained or stored. Thus, if the input address of some device is #6800 (hexadecimal), then the statement `LET X=@#6800` inputs from the device, and, if the output address is #6801, then `LET @#6801=Y` outputs to the device. As another example, if it is desired to store a matrix of dimension M x N starting at a location stored in A, then the (I, J)th byte of this matrix may be accessed as follows:

```
LET @ (A+ N*(I-1) + (J-1)) = X
```

if the array is to be stored by rows and indexed from (1, 1). If indexing from (0, 0) is acceptable, as is standard in larger BASIC's, and the matrix is to be stored by columns, then the proper assignment is as follows:

```
LET @ (A+I+(J*M)) = X
```

NOTE

> The Indirect Operator allows one to access memory locations only one byte at a time. Thus, the matrix defined in the example above is a matrix of one-byte elements. An assignment such as `LET @A=256` would actually result in changing the memory location pointed to by A to zero, not 256, because only the least significant byte of 256 (which is zero) is stored at that location.

5.2 INDIRECT OPERATOR SUMMARY

- Any place a variable, as an example: V, would be legal, the construct "@V" is also legal. The meaning of "@V" is the byte located at the memory location whose address is the value of V. Thus, if the value of V is 10, then `LET @V=100` stores a decimal 100 in byte 10 and `LET W=@V` then sets W to 100.
- Also legal:

```
PRINT   @10
LET     E=@(A+10*I+J)
```

NOTE

> Parentheses are required when applying @ to an expression.

Chapter 6

MULTIPLE STATEMENTS, PAGES, AND STRING HANDLING

6.1 PROGRAM LINE MULTIPLE STATEMENTS

More than one NIBL statement can be placed on a program line. This is accomplished by placing a colon (':') between the statements to separate them. This practice can improve the readability of programs, and can reduce the amount of memory required to store the program. As an example of the use of multiple statements on a line, consider the following NIBL line:

```
100 PR "FUEL =",X: IF X< 0 PR "YOU BLEW IT": GOTO 32767
```

The first Output (PRINT) Statement is executed; then, if X is negative, the "YOU BLEW IT!" message is printed, and NIBL proceeds to the last line in the program (if there is a line 32767). If X had been positive, however, the last message would not have been printed, and control would have passed immediately to the next numbered line in the program after line 100.

The above example demonstrates that if the condition in an IF Statement fails, control is transferred to the next program line, and anything else on the line is ignored by NIBL.

6.2 MULTIPLE PAGE HANDLING

6.2.1 4K PAGES

The NIBL system requires RAM from 1000 to 17FF (hex); the RAM, together with the optional RAM from 1800 to 1FFF, comprises PAGE 1. This 4K Page is normally used to store a program; however, NIBL also allows the user to add up to 6 more 4K Pages for storing other programs. These Pages are called PAGES 2-7. This section describes briefly how to make use of this hardware-dependent feature of NIBL.

6.2.2 Transferring PAGE Control

The pseudo-variable PAGE refers to the 4K Page in which a NIBL program is currently being executed or edited. Legal Pages are 1-7; any assignment to PAGE (as an example: PAGE = PAGE + 1) uses only the least significant 3 bits of the value as the new page number. PAGE 0 is the same as PAGE 1. An assignment to PAGE during program execution causes control to be transferred to the first program line in that PAGE, if any. If control is transferred from one Page to another by some other means (that is, via RETURN, UNTIL, or NEXT Statements), the correct value of PAGE is automatically updated by NIBL. Page selection allows seven completely separate programs to be maintained by the user; however, all programs use the same variables and stacks.

6.2.3 PAGES 2 Through 7

PAGES 2 through 7 (corresponding to Starting Addresses 2000 to 7000 (hex)) may be ROM. When NIBL is run initially, it attempts to initialize all pages, then transfers control to PAGE 2. If PAGE 2 does not contain a NIBL program, the value of PAGE will be set to 1, and NIBL will immediately prompt with a (">") GREATER THAN sign at the Teletype. If PAGE 2 contains a NIBL program in the proper format, it will be executed immediately without the need for the user to tell NIBL anything.

6.2.4 Moving NIBL Program Into ROM

Here is how to move a NIBL program into a PROM, once it has been written and entered to the NIBL system (assuming the user has an LCDS). The final address of the program is found by typing `PRINT TOP`. Convert the number that NIBL prints into hex for the LCDS. Halt the LCDS, and punch the section of memory containing the program in Load Module Format. PAGE 1 programs begin at location 11lE (hex); all other Pages start at n000, where n is the page number. Once the Load Module is punched, use SPPRO on the PAGE DOS to program some PROMs of the users NIBL program. Such a program can be moved from Page to Page, if so desired.

6.3 STRING HANDLING

6.3.1 String Input

String input is accomplished by executing a statement of the form `INPUT$ F`, where F is a factor, syntactically; refer to Appendix A. NIBL prompts with `?`, indicating that the user is to type in a line terminated by (CR), the Carriage Return Key. All line editing characters can be used (Backspace, Line Delete, and so on). The line is stored in consecutive locations starting at the address stored in F, up to and including the (CR).

6.3.2 String Output

An item in an Output (PRINT) Statement can include the form `$F`, where F is a factor. This will cause the string beginning at the address F to be printed, up to, but not including, the (CR). A Keyboard Interrupt will also terminate the printing if detected before the (CR).

6.3.3 String Assignment

A statement of the form `$F = "THIS IS A STRING".` causes the characters in quotes to be stored in memory starting at the address indicated by F (which is again a factor), with the (CR) appended to the string.

6.3.4 String Move

A statement of the form `$F = $G`, where F and G are factors, causes the characters in memory beginning with the Address G to be transferred byte-by-byte to memory starting at Address F, until a carriage return is detected in the source string, or a Keyboard Interrupt is detected. The last character, normally a carriage return, is also copied. A statement such as `$(F+1) = $F` is disastrous unless the user presses the BREAK Key very quickly, since it causes the entire contents of the Page pointed to by F to be filled with the first character of $F.

Page wraparound will occur if the user is not careful, but F and G may be in different Pages.

An example of a program using String Move and Input Statements follows; strings are typically put at the top of a program in memory:

```
: NIBL code
:
:
;
Q = TOP:        REM MAKES Q POINT TO EMPTY RAM ABOVE TOP OF PROGRAM
R = TOP + 10:   REM MAKES R POINT TO RAM 10 BYTES ABOVE Q
:
:
$Q = "A STRING"
INPUT $R
:
:
```

6.3.5 String Handling Summary

- $T = "THIS IS A STRING"
- PRINT $T, $(TOP+72)
- INPUT $ (U+20)
- $U = $(TOP + 2*36)

Chapter 7

ADDITIONAL STATEMENTS

7.1 LINK STATEMENT

LINK <address> causes control to be transferred to the SC/MP Machine Language Routine starting at <address>. Control is transferred by execution of an XPPC P3 instruction. The Machine Language Routine should make sure that P3 is restored to its original value in order to return to NIBL (by execution of another XPPC P3 instruction). The other pointers may be modified by the routine. P1's value is unpredictable; P2 points to the start of A-Z variable storage. Variables are stored in alphabetically ascending order; two bytes each; low-order byte, then high-order byte.

7.2 REMARK STATEMENT

The REM Statement is used to insert comments into NIBL programs. When NIBL encounters the word REM as the first word in a statement, it skips to the next line in the program. It is wise to insert many comments into a program while it is being developed. After the program is debugged, the comments can be removed if the user requires more space and speed. However, if someone else besides the original programmer must understand or modify the program someday, those comments can be extremely helpful.

7.3 END STATEMENT

The END Statement is useful for inserting breakpoints into a NIBL program while it is being debugged. When NIBL encounters an END statement, it prints a break message and the current line number, and returns to edit mode. Any number of END statements may appear in a program, and the last line in the program does not need to be an END Statement.

Appendix A

NIBL FORMAL GRAMMAR

All items in single quotes are actual symbols in NIBL; all other identifiers are symbols in the grammar. The equals sign "=", means "is defined as"; parentheses are used to group several items together as one item; the exclamation point, "!", means an exclusive-or choice between the items on either side of it; the asterisk, "*", means zero or more occurrences of the item to its left; the plus sign, "+", means one or more repetitions; the question mark, "?", means zero or one occurrence; and the semicolon, ";", marks the end of a definition.

```
Nibl-line = Immediate-statement
          ! Program-line
          ;

Immediate-statement = (Command ! Statement) Carriage-return;

Program-line = (Decimal-number Statement-list Carriage-return) ;

Command = 'NEW' Decimal-number ?
          ! 'CLEAR'
          ! 'LIST' Decimal-number ?
          ! 'RUN'
          ;

Statement-list = Statement (':' Statement) *;

Statement = 'LET' ? Left-part '=' Rel-exp
          ! 'LET' ? '$' Factor '=' (String ! '$' Factor)
          ! 'GO' ('TO' ! 'SUB') Rel-exp
          ! 'RETURN'
          ! ('PR' ! 'PRINT') Print-list
          ! 'IF' Rel-expr 'THEN' ? Statement
          ! 'DO'
          ! 'UNTIL' Rel-exp
          ! 'FOR' Variable '=' Rel-exp 'TO' Rel-exp ('STEP' Rel-exp) ?
          ! 'NEXT' Variable
          ! 'INPUT' (Variable + ! '$' Factor)
          ! 'LINK' Rel-exp
          ! 'REM' Any-Character-Except-Carriage-Return *
          ! 'END'
          ;

Left-part = (Variable ! '@' Factor ! 'STAT' ! 'PAGE') ;

Rel-exp   =  Expression Relop Expression
          !  Expression
          ;

Relop  =  '<' ! '<' '=' ! '<' '>' ! '>' ! '>' '=' ! '=' ;

Expression = Expression Adding-operator Term
           ! ('+' ! '-') ? Term
           ;
```

(Continued on Page A-2)

```
Adding-operator = '+' ! '-' ! 'OR' ;
```

```
Term  = Term Multiplying-operator Factor
      ! Factor
      ;
```

```
Multiplying-operator = '*' ! '/' ! 'AND' ;
```

```
Factor  = Variable
        ! Decimal-number
        ! '(' Rel-exp ')'
        ! '@' Factor
        ! '#' Hex-number
        ! 'NOT' Factor
        ! 'MOD' '(' Rel-exp ',' Rel-exp ')'
        ! 'RND' '(' Rel-exp ',' Rel-exp ')'
        ! 'STAT'
        ! 'TOP'
        ! 'PAGE'
        ;
```

```
Variable = 'A' ! 'B' ! 'C' ! ... ! 'Y' ! 'Z' ;
```

```
Decimal-number = Decimal-digit + ;
```

```
Decimal-digit = '0' ! '1' ! '2' ! ... ! '9' ;
```

```
Hex-number = (Decimal-digit ! Hex-digit) + ;
```

```
Hex-digit = 'A' ! 'B' ! 'C' ! 'D' ! 'E' ! 'F' ;
```

```
Print-list = Print-item (',' Print-item) * ;
```

```
Print-item = (String ! Rel-exp ! '$' Factor) ;
```

```
String = '"' Almost-Any-Character '"' ;
```

Spaces are not usually significant in a NIBL program, with the following exceptions: Spaces cannot appear within key words (like 'THEN' or 'PRINT') or within constants. Also, a variable (like 'A' or 'Z') must be followed immediately by a non-alphabetic character to distinguish it from a key word.

Appendix B

NIBL ERROR MESSAGES AND DESCRIPTIONS

Error messages are of the form:

EEEE ERROR AT LN

where EEEE is one of the error codes below, and LN is the number of the line in which the error was encountered.

CHAR	Character after logical end of statement
DIV0	Division by zero
END"	No ending quote on string
FOR	FOR without NEXT
NEST	Nesting limit exceeded in expression, FOR's, GOSUBs, etc.
NEXT	NEXT without FOR
NOGO	No line number corresponding to GOTO or GOSUB
RTRN	RETURN without previous GOSUB
SNTX	Syntax error
STMT	Statement type used improperly
UNTL	UNTIL without DO
VALU	Constant format or value error
AREA	No more room left in current page for program

Appendix C

"BAGELS" — A SIMPLE NIBL GAME

The object of the game is to guess the number that the microprocessor has picked. All numbers are between 100 and 999. For each correctly guessed digit in the correct location, the processor responds "FERMI". For each correct digit not in the right location, the processor responds "PICO". If no correct digits are guessed, the processor responds "BAGELS".

```
    ;
10  PRINT"                    BAGELS": PRINT"": PRINT""
40  PRINT" I WILL THINK OF A THREE DIGIT NUMBER.  YOU TRY TO"
50  PRINT" GUESS WHAT IT IS.  FOR EACH CORRECT DIGIT IN THE"
53  PRINT" CORRECT LOCATION, I WILL PRINT 'FERMI'.  FOR EACH"
55  PRINT" CORRECT DIGIT NOT IN THE CORRECT LOCATION, I WILL"
57  PRINT" PRINT 'PICO'.  IF NO DIGITS ARE CORRECT, I WILL PRINT"
58  PRINT" 'BAGELS'.": PRINT"": PRINT""
59  REM
60  A=RND(1,9):B=RND(0,9):C=RND(0,9):P=0
70                            REM SELECT A NUMBER
71  REM
120 PRINT "PLEASE GUESS A THREE DIGIT NUMBER. ";
130 INPUT G:                  REM INPUT GUESS, TEST RANGE
135 REM
140 IF G >1000 OR G< 100 GOTO 120
160 M=0: N=0: P=P+1: H=G/100: REM ZERO CNTRS, SELECT LEFT DIGIT
200 IF H=A M=M+1:             REM CORRECT DIGIT & LOCATION
210 IF ((H=B)OR(H=C)) N=N+1:  REM CORRECT DIGIT, BAD LOCATION
230 I=MOD(G,100)/10:          REM SELECT MID. DIGIT OF INPUT
240 IF ((I=A)OR(I=C)) N=N+1:  REM CORRECT DIGIT, BAD LOCATION
250 IF I=B M=M+1:             REM CORRECT DIGIT & LOCATION
270 J=MOD(G,10):              REM SELECT RIGHT DIGIT OF INPUT
280 IF ((J=A)OR(J=B)) N=N+1:  REM CORRECT DIGIT, BAD LOCATION
300 IF J=C M=M+1:             REM CORRECT DIGIT & LOCATION
310 IF M< 3 GOTO 600
320 PRINT"CONGRATULATIONS! YOU GOT IT IN",P,"TRIES."
330 PRINT" PLAY AGAIN? (1=YES, 0=NO)"
340 INPUT Q: IF Q=0 GOTO 1000
360 GOTO 60
500 REM
550                           REM NEXT SECTION PRINTS CLUES
600 IF M< >0 FOR T=1 TO M:PRINT"FERMI ";:NEXT T
620 IF N< >0 FOR T=1 TO N:PRINT"PICO  ";:NEXT T
650 IF M+N=0 PRINT "BAGELS"
700 PRINT"":GOTO 120:         REM ASK FOR NEXT GUESS
1000 PRINT"GOODBYE"
```