THE "NIBBLER"

DIGI-KEY CORPORATION
THIEF RIVER FALLS, MN. 56701
(218) 681-6674

TABLE OF CONTENTS

PAGE

1. INTRODUCTION ---- 1

2. NIBBLER SOFTWARE GUIDE ---- 3

2.1 ADDITIONS AND CORRECTIONS TO SC/MP NIBL REFERENCE GUIDE ---- 4
2.2 SC/MP NIBL REFERENCE GUIDE ---- 5
2.3 NIBL ROM LISTING WITH CHANGES FOR N-CHANNEL 4 MHZ SC/MP II ---- 36
2.4 SHORT PROGRAM EXAMPLES ---- 91

3. NIBBLER HARDWARE GUIDE ---- 96

3.1 THEORY OF OPERATION ---- 97
3.1.1 LOGIC CONVENTION ---- 97
3.1.2 DISCUSSION OF SIGNALS ---- 98
3.1.3 DETAILED DISCUSSION OF MEMORY ADDRESSING ---- 101

3.2 TYPICAL SYSTEM CONFIGURATION ---- 103

3.3 BACKPLANE DESCRIPTION ---- 106

3.4 BOARD JUMPERING ---- 107

3.5 INTERFACING ---- 108
3.5.1 POWER INTERFACING ---- 108
3.5.2 TELETYPE INTERFACING ---- 109
3.5.3 A SAMPLE I/O APPLICATION DISCUSSION ---- 113
3.5.4 A LOOK AT THE EASE OF I/O PROGRAMMING ---- 115

3.6 LOGIC DIAGRAMS ---- 119
3.6.1 BLOCK DIAGRAM ---- 120
3.6.2 BOARD AND EDGE CONNECTOR MAP ---- 121
3.6.3 MAIN LOGIC ---- 122
3.6.4 ROM LOGIC ---- 123
3.6.5 RAM LOGIC ---- 124
3.6.6 WIRE LIST A CHIPS ---- 125
3.6.7 WIRE LIST B CHIPS ---- 127
3.6.8 WIRE LIST C CHIPS ---- 129
3.6.9 WIRE LIST LARGE SIGNAL NETS ---- 131

3.7 CHIP TECHNICAL DESCRIPTIONS ---- 132
3.7.1 NIBL ROMS ---- 133
3.7.2 SC/MP II ---- 137

3.8 NIBBLER SUPPORT HARDWARE ---- 142
3.8.1 EDGE CONNECTOR ---- $5.00* ---- 143
3.8.2 CARD FILE ---- $39.95* ---- 143
3.8.3 EXTENDER CARD ---- $29.95* ---- 143
3.8.4 PROTOTYPING CARD ---- $39.95* ---- 143
3.8.5 NIBBLER POWER SUPPLY AND INTERFACING BOARD - $39.95* ---- 144

4. SUPPORT DOCUMENTATION ---------- 145

4.1 INTERFACE AGE REPRINT ---------- 146

4.2 BIBLIOGRAPHY OF SUPPORTING DOCUMENTATION ---------- 154

4.2.1 SC/MP INSTRUCTION GUIDE	$1.00*
4.2.2 SC/MP TECHNICAL DESCRIPTION	$5.00*
4.2.3 SC/MP MICROPROCESSOR APPLICATIONS HANDBOOK	$5.00*
4.2.4 SC/MP MICROPROCESSOR ASSEMBLY LANGUAGE	$10.00*

5. IF YOUR NIBBLER DOESN'T WORK---------- 155

*THESE ITEMS AVAILABLE AT EXTRA COST FROM DIGI-KEY

1. INTRODUCTION

THE NIBBLER IS A SINGLE BOARD MICROCOMPUTER DESIGNED TO BE A LOW COST HOBBY COMPUTER OR INTELLIGENT CONTROLLER FOR INDUSTRIAL APPLICATIONS. THE NIBBLER IS DESIGNED AROUND NATIONAL SEMICONDUCTOR'S SC/MP II MICROPROCESSOR AND NIBL (PRONOUNDED NIBBLE) BASIC ROM CHIPS. THE SC/MP II MICROPROCESSOR UTILIZES THE NIBL ROM TO PROVIDE AN EXTENDED VERSION OF TINY BASIC.

TINY BASIC IS A SIMPLIFIED VERSION OF THE COMPUTER LANGUAGE BASIC, DEVELOPED BY DR. JOHN KEMENY AND DR. THOMAS KURTZ AT DARTMOUTH COLLEGE IN 1963. BASIC HAS BECOME THE "PEOPLE'S COMPUTER LANGAUGE" BECAUSE IT IS EASY TO LEARN AND EASY TO USE BY PEOPLE WHO ARE NOT COMPUTER SCIENTISTS OR PROFESSIONAL PROGRAMMERS. THE USERS OF BASIC ARE ENGINEERS, TECHNICIANS, SCIENTISTS, STATISTICIANS, BUSINESS PEOPLE, HOBBIESTS, TEACHERS, COLLEGE STUDENTS, AND A VAST MULTITUDE OF STUDENTS IN ELEMENTARY AND SECONDARY SCHOOLS.

THE ORIGINAL TINY BASIC WAS DESIGNED FOR APPLICATIONS SUCH AS INTEGER ARITHMETIC PROGLEMS, COMPUTER GAMES AND TEACHING BEGINNERS HOW TO PROGRAM COMPUTERS. NIBL HAS EXTENDED CAPABILITIES WHICH MAKE IT A POWERFUL DESIGN TOOL FOR DEVELOPING CONTROL APPLICATIONS, AS WELL AS THE APPLICATIONS FOR WHICH TINY BASIC WAS ORIGINALLY DESIGNED.

INFORMATION ON TINY BASIC WAS FIRST PUBLISHED IN "PEOPLE'S COMPUTER COMPANY, VOL 3, NO 4 (MARCH 1975) AND VOL 4, NO 1 (JULY 1975). HOWEVER THE BEST SOURCE OF INFORMATION ON TINY BASIC IS DR. DOBB'S JOURNAL OF COMPUTER CALISTHENICS AND ORTHODONTIA, BEGINNING WITH VOL 1, NO 1 (JAN 1976) AND CONTINUING THROUGH SEVERAL ISSUES.

THE SC/MP II WAS CHOSEN BY DIGI-KEY FOR ITS LOW COST AND THE AVAILABILITY OF A TESTED AND PROVEN BASIC INTERPRETER, THE LARGE NUMBER OF SC/MP II USERS AND A GOOD SUPPLY OF DOCUMENTATION SUPPORTING SC/MP APPLICATIONS. DIGI-KEY IS PROVIDING THE HOBBIEST AS WELL AS THE INDUSTRIAL USER A LOW COST MODULE THAT CAN EASILY BE USED AS AN INTRODUCTION INTO THE WORLD OF MICROCOMPUTING.

THE LARGEST PROBLEM FACING A BEGINNER IN MICROCOMPUTING IS THE SEEMINGLY OVERWHELMING AMOUNT OF INFORMATION THAT MUST BE ASSIMILATED BEFORE HE CAN GET STARTED. MOST LOW COST SYSTEMS REQUIRE THE USER TO LEARN THE MACHINE LANGUAGE OF THE MICROPROCESSOR BEFORE HE CAN PROCEED. ON THE OTHER HAND, IF A HIGHER LEVEL LANGUAGE IS AVAILABLE SUCH AS BASIC, THEN THE COST IS INCREASED. DIGI-KEY IS PROVIDING THE NIBBLER AS A SOLUTION TO THIS PROBLEM. THE NIBBLER WITH A COST OF $149.95 PROVIDES ENOUGH COMPUTER POWER TO ALLOW IMMEDIATE EDUCATIONAL PROGRESS AT A REMARKABLE RATE. BASIC IS EASY TO LEARN AND USE. NIBL PROVIDES ENOUGH EXTRA POWER TO ALLOW ACCESS TO THE FUNDAMENTAL FUNCTIONS OF THE MICROPROCESSOR FOR INDUSTRIAL APPLICATIONS. THE INDIRECT OPERATOR ALLOWS MEMORY READS AND WRITES JUST LIKE MACHINE LANGUAGE. THE STRING MANIPULATION CAPABILITY ALLOWS BYTE WISE MANIPULATION OF DATA. THE LOGICAL OPERATORS ALLOW BIT MANIPULATION.

THE NIBBLER OFFERS INTEGER ARITHMETIC. ALL CALCULATIONS ARE PERFORMED USING INTEGERS. THE LARGEST NUMBER THAT CAN BE REPRESENTED IN AN INTEGER BASIC IS A FUNCTION OF THE STORAGE SPACE AVAILABLE FOR THE NUMBER. NIBL USES TWO BYTES (16 BITS) FOR NUMBER STORAGE. 15 BITS ARE USED FOR THE MAGNITUDE OF THE NUMBER AND 1 BIT FOR THE SIGN. THEREFORE THE LARGEST NUMBER REPRESENTED IN NIBL IS 2 RAISED TO THE 15TH POWER LESS 1 OR 32767. THE RESULT OF ANY ARITHMETIC OPERATION WILL ALSO BE AN INTEGER, FOR EXAMPLE 3/2=1. THE PROBLEM HERE IS OBVIOUS, WHERE IS THE REMAINDER? THE SOLUTION IS THE USE OF THE MOD INSTRUCTION. MOD STANDS FOR MODULO ARITHMETIC AND ALLOWS YOU TO FIND REMAINDERS OF DIVISION DIRECTLY. THEREFORE IT IS POSSIBLE TO WRITE ROUTINES THAT CAN PERFORM FLOATING POINT ARITHMETIC IF YOU DESIRE TO HAVE THIS CAPABILITY. THE NIBBLER CAN BE USED FOR AN ENORMOUS VARIETY OF FUNCTIONS WITHOUT THE FLOATING POINT CAPABILITY. SEVERAL REFERENCE BOOKS ARE AVAILABLE FROM DIGI-KEY TO ASSIST YOU IN UNDERSTANDING HOW TO UTILIZE YOUR NIBBLER IN A WIDE NUMBER OF APPLICATIONS.

2. "NIBBLER SOFTWARE GUIDE"

2.1 ADDITIONS AND CORRECTIONS FOR SC/MP NIBL REFERENCE GUIDE

THE SC/MP NIBL REFERENCE GUIDE IS PROVIDED AS A SOFTWARE MANUAL. THERE ARE SEVERAL ADDITIONS AND CORRECTIONS WHICH ARE NOTED BELOW. THE COMMENTS ARE ORGANIZED BY THE EXISTING HEADING NUMBERS.

1.1 INTRODUCTION:	THE NIBBLER USES THE SC/MP II MICROPROCESSOR
FIGURE 1-2:	PLEASE UTILIZE SHEET 1 OF THE NIBBLER LOGIC FOR THE BLOCK DIAGRAM.
1.3.1.E:	A CRT TERMINAL SUCH AS A MICROTERM ACT 1A CAN BE USED WITH THE TTL INTERFACE.
1.3.1.F:	ONLY A 5 VOLT 0.4 AMP SUPPLY IS REQUIRED IF NOT USING A TELETYPE.
6.2.4:	THE NIBBLER CAN BE USED WITH A ROM PROGRAM AS DESCRIBED IN THE HARDWARE DESCRIPTION. HOWEVER THE NIBBLER DOES NOT SUPPORT THE LCDS.
APPENDIX B:	THE "AREA" ERROR MESSAGE WILL NOT RESULT WHEN MEMORY SIZE IS EXCEEDED WITH A 2K RAM SYSTEM. THE NIBBLER WILL LOCK UP AND HAVE TO BE RESET.
APPENDIX C:	THE BAGLES PROGRAM STATEMENT 140 SHOULD READ: "140 IF (G>1000) OR (G<100) GOTO 120".

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

Chapter			Page
1	GENERAL INFORMATION		
	1.1	INTRODUCTION	1-1
	1.2	SYSTEM DESCRIPTION	1-1
	1.3	HARDWARE CONFIGURATION	1-1
	1.3.1	Minimum Hardware Configuration	1-1
	1.3.2	Additional Memory	1-2
	1.3.3	Using a SC/MP-II	1-2
2	PROGRAM ENTRY AND CONTROL		
	2.1	PROGRAM ENTRY	2-1
	2.1.1	Prior to Program Entry	2-1
	2.1.2	Entering Program Lines	2-1
	2.1.3	Punctuation with Blanks	2-1
	2.1.4	Additional Proper Procedures	2-1
	2.1.5	Control Characters	2-1
	2.1.6	Program Entry Summary	2-2
	2.2	PROGRAM CONTROL	2-2
	2.2.1	NIBL Commands	2-2
	2.2.1.1	NEW Command	2-2
	2.2.1.2	RUN and GOTO Commands	2-2
	2.2.1.3	LIST Command	2-3
	2.2.1.4	CLEAR Command	2-3
	2.2.2	Program Control Summary	2-3
3	NIBL EXPRESSIONS AND FUNCTIONS		
	3.1	EXPRESSIONS.	3-1
	3.1.1	Variable Names	3-1
	3.1.2	Relational Operators	3-1
	3.1.3	Arithmetic Operators	3-1
	3.1.4	Logical Operators.	3-1
	3.1.5	Expression Summary.	3-1
	3.2	FUNCTIONS	3-1
	3.2.1	MOD Function	3-2
	3.2.2	RND Function	3-2
	3.2.3	STAT Function	3-2
	3.2.4	TOP Function	3-2
	3.2.5	PAGE Function	3-2
	3.2.6	Function Summary	3-2
	3.3	HIERARCHY OF OPERATIONS	3-3
4	INPUT/OUTPUT, ASSIGNMENT, AND CONTROL STATEMENTS		
	4.1	INPUT/OUTPUT STATEMENTS	4-1
	4.1.1	Executing an Input/Output Statement	4-1
	4.1.2	Output Statements.	4-1
	4.1.3	Input/Output Statement Summary	4-1
	4.2	ASSIGNMENT STATEMENTS	4-1
	4.2.1	Assignment Statement Forms	4-1
	4.2.2	Assignment Statement Summary	4-2
	4.3	CONTROL STATEMENTS	4-2
	4.3.1	Control Statement Construction Allowances	4-2
	4.3.2	Subroutine Advantages	4-3
	4.3.3	DO and UNTIL Statements	4-3
	4.3.4	FOR and NEXT Statements	4-3
	4.3.5	Control Statement Summary.	4-4

TABLE OF CONTENTS (Continued)

Chapter			Page
5	THE INDIRECT OPERATOR		
	5.1	INDIRECT OPERATOR USAGE	5-1
	5.2	INDIRECT OPERATOR SUMMARY	5-1
6	MULTIPLE STATEMENTS, PAGES, AND STRING HANDLING		
	6.1	PROGRAM LINE MULTIPLE STATEMENTS	6-1
	6.2	MULTIPLE PAGE HANDLING	6-1
	6.2.1	4K PAGES	6-1
	6.2.2	Transferring PAGE Control	6-1
	6.2.3	PAGES 2 Through 7	6-1
	6.2.4	Moving NIBL Program Into ROM	6-2
	6.3	STRING HANDLING	6-2
	6.3.1	String Input	6-2
	6.3.2	String Output	6-2
	6.3.3	String Assignment	6-2
	6.3.4	String Move	6-2
	6.3.5	String Handling Summary	6-3
7	ADDITIONAL STATEMENTS		
	7.1	LINK STATEMENT	7-1
	7.2	REMARK STATEMENT	7-1
	7.3	END STATEMENT	7-1
APPENDIX A	NIBL FORMAL GRAMMAR		
APPENDIX B	NIBL ERROR MESSAGES AND DESCRIPTIONS		
APPENDIX C	"BAGELS" — A SIMPLE NIBL GAME		

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.

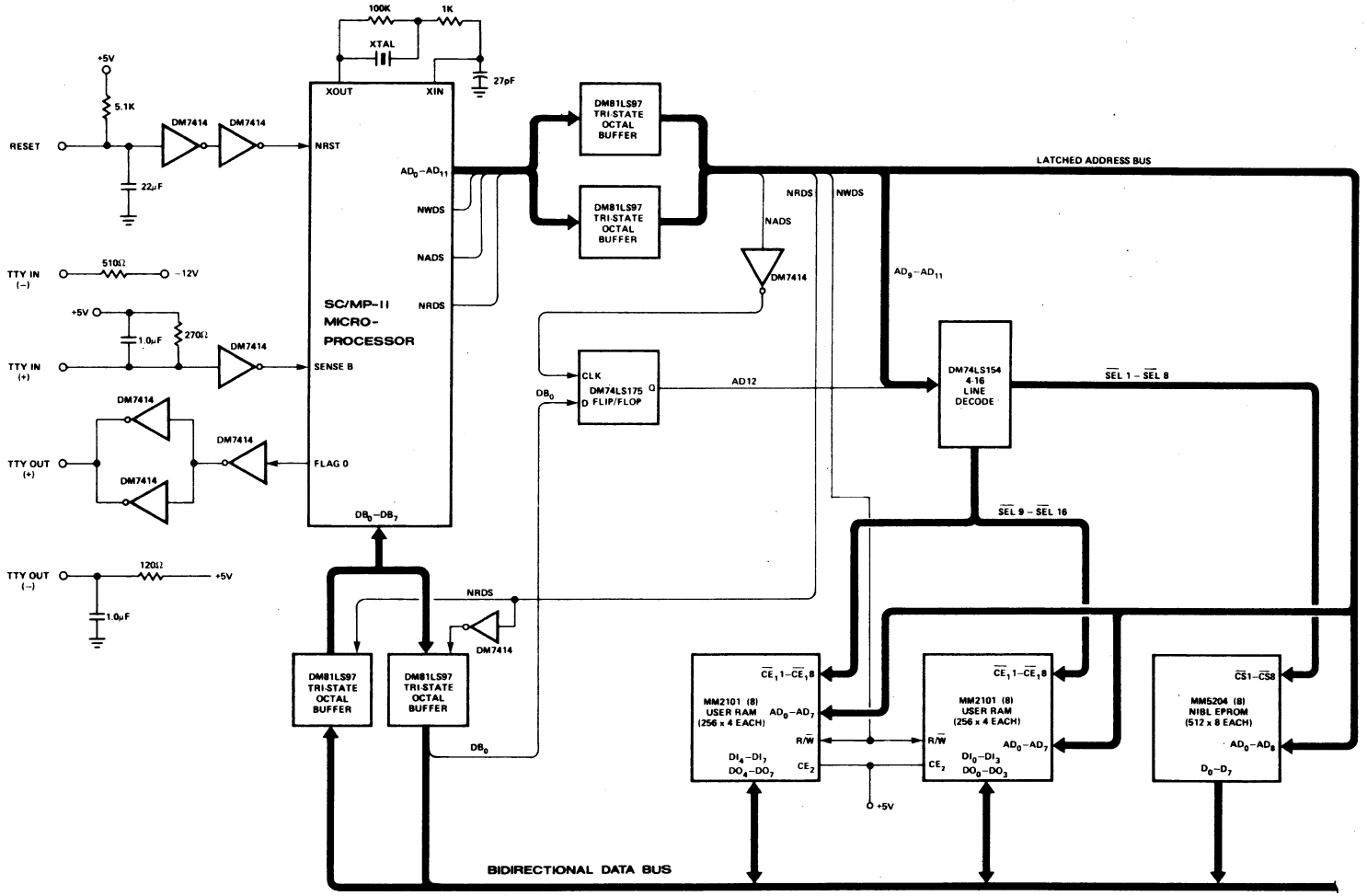


Figure 1-1. Minimum Hardware Configuration to Support the NIBL Interpreter (ISP-8F/351)

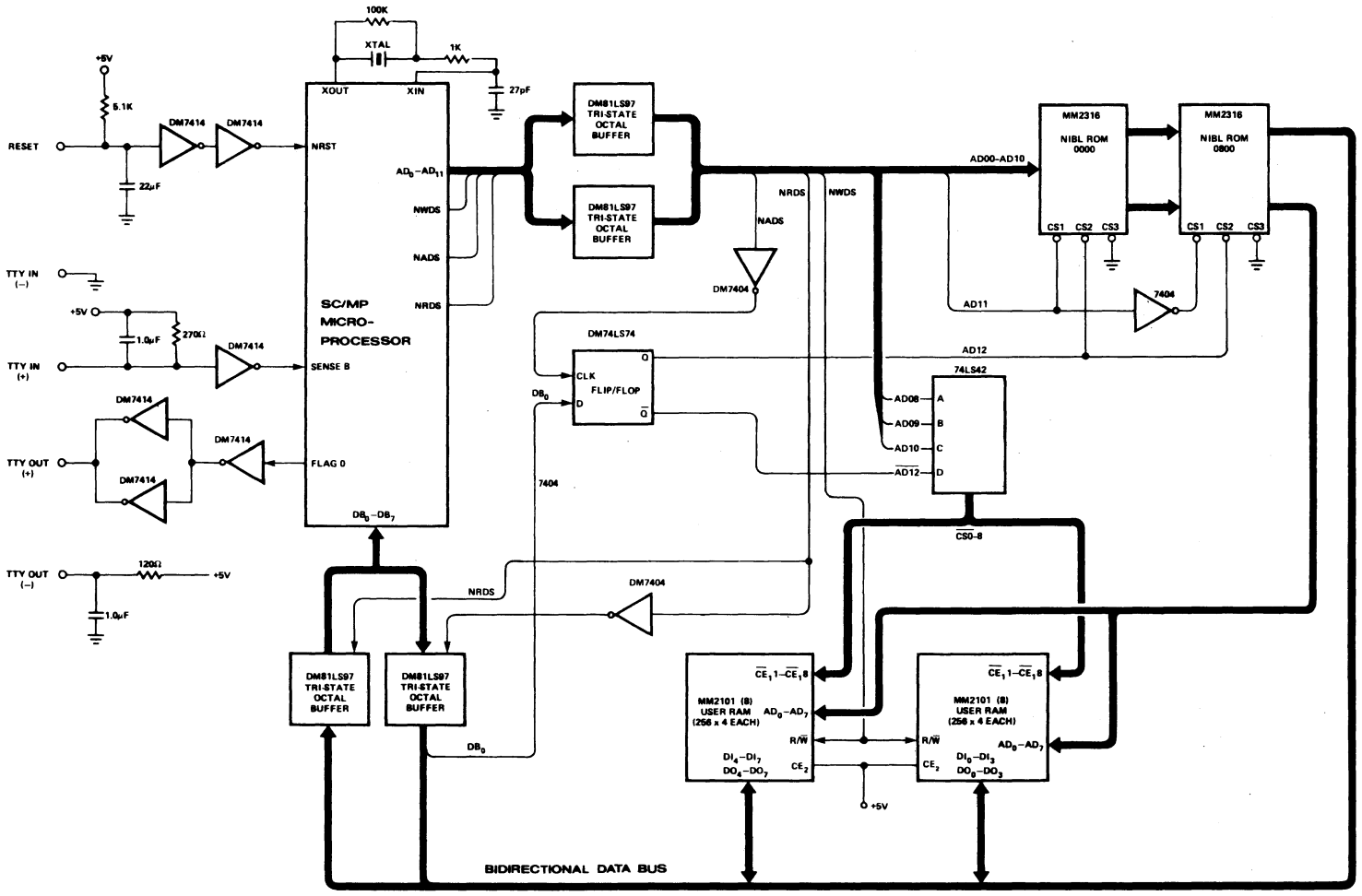


Figure 1-2. Minimum Hardware Configuration to Support the NIBL Interpreter (ISP-8F/352)

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 111E (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"
```

2.3 NIBL ROM LISTING

THE ROM LISTING IS FOR THE P-CHANNEL SC/MP I. THE CHANGES FOR THE SC/MP II (TTY ROUTINES) HAVE BEEN ANNOTATED.

NATIONAL SEMICONDUCTOR

P-SC/MP

CROSS ASSEMBLER

TITLE: NIBL 12/17/76

p-channel version

SC/MP-I

NATIONAL SEMICONDUCTOR
WESTERN MICROPROCESSOR TRAINING CENTER
1333 LAWRENCE EXPWY., SUITE 430
SANTA CLARA, CA 95051
TEL (408) 247-7924

```
                        .TITLE  NIBL,'12/17/76'
0001                    .LIST   1

                ;*************************************************************
                ;*                                                           *
                ;*      WE ARE TIED DOWN TO A LANGUAGE WHICH                 *
                ;*      MAKES UP IN OBSCURITY WHAT IT LACKS                  *
                ;*      IN STYLE.                                            *
                ;*                          -- TOM STOPPARD                  *
                ;*                                                           *
                ;*************************************************************

0020    TSTBIT  =       020             ;I.L.  INSTRUCTION FLAGS
0040    JMPBIT  =       040
0080    CALBIT  =       080
0001    P1      =       1               ;SC/MP POINTER ASSIGNMENTS
0002    P2      =       2
0003    P3      =       3
FF80    EREG    =       -128            ;THE EXTENSION REGISTER

                ;  DISPLACEMENTS FOR RAM VARIABLES USED BY INTERPRETER

FFFF    DOPTR   =       -1              ;DO-STACK POINTER
FFFE    FORPTR  =       -2              ;FOR-STACK POINTER
FFFD    LSTK    =       -3              ;ARITHMETIC STACK POINTER
FFFC    SBRPTR  =       -4              ;GOSUB STACK POINTER
FFFB    PCLOW   =       -5              ;I.L.  PROGRAM COUNTER
FFFA    PCHIGH  =       -6
FFF9    PCSTK   =       -7              ;I.L.  CALL STACK POINTER
FFF8    LOLINE  =       -8              ;CURRENT LINE NUMBER
FFF7    HILINE  =       -9
FFF6    PAGE    =       -10             ;VALUE OF CURRENT PAGE
FFF5    LISTNG  =       -11             ;LISTING FLAG
FFF4    RUNMOD  =       -12             ;RUN/EDIT FLAG
FFF3    LABLLO  =       -13
FFF2    LABLHI  =       -14
FFF1    P1LOW   =       -15             ;SPACE TO SAVE CURSOR
FFF0    P1HIGH  =       -16
FFEF    LO      =       -17
FFEE    HI      =       -18
FFED    FAILLO  =       -19
FFEC    FAILHI  =       -20
FFEB    NUM     =       -21
FFEA    TEMP    =       -22
FFE9    TEMP2   =       -23
FFE8    TEMP3   =       -24
FFE7    CHRNUM  =       -25
FFE6    RNDF    =       -26
FFE5    RNDX    =       -27             ;SEEDS FOR RANDOM NUMBER
FFE4    RNDY    =       -28

                ;  ALLOCATION OF RAM FOR NIBL VARIABLES, STACKS,
                ;    AND LINE BUFFER
```

```
 55 0000                .=01000+28
 56 101C        VARS:   .=.+52                  ;NIBL VARIABLES A-Z
 57 1050        AESTK:  .=.+26                  ;ARITHMETIC STACK
 58 106A        SBRSTK: .=.+16                  ;GOSUB STACK
 59 107A        DOSTAK: .=.+16                  ;DO/UNTIL STACK
 60 108A        FORSTK: .=.+28                  ;FOR/NEXT STACK
 61 10A6        PCSTAK: .=.+48                  ;I.L. CALL STACK
 62 10D6        LBUF:   .=.+74                  ;LINE BUFFER
 63 1120        PGM:    .=0                     ;USER'S PROGRAM
 64
 65                     .MACRO  LDPI,P,VAL
 66                     .MLOC   TEMP
 67                     .SET    TEMP,VAL
 68                     LDI     H(TEMP)
 69                     XPAH    P
 70                     LDI     L(TEMP)
 71                     XPAL    P
 72                     .ENDM
 73
 74
 75             ;****************************************
 76             ;*       INITIALIZATION OF NIBL         *
 77             ;****************************************
 78
 79
 80 0000 08             NOP
 81 0001                LDPI    P2,VARS         ;POINT P2 AT VARIABLES
 82 0007                LDPI    P1,PGM          ;POINT P1 AT PAGE ONE PROGRAM
 83 000D C4FF           LDI     -1              ;STORE -1 AT START OF PROGRAM
 84 000F C900           ST      0(P1)
 85 0011 C901           ST      1(P1)
 86 0013 C40D           LDI     0D              ;ALSO STORE A DUMMY
 87 0015 C9FF           ST      -1(P1)          ;  CARRIAGE RETURN
 88 0017 C402           LDI     2               ;POINT P2 AT PAGE 2,
 89 0019 CAF6           ST      PAGE(P2)        ; INITIALLY SET PAGE TO 2
 90 001B 31             XPAL    P1
 91 001C C420           LDI     020
 92 001E 35             XPAH    P1
 93 001F B902           DLD     2(P1)           ;CHECK IF THERE IS REALLY
 94 0021 01             XAE                     ; A PROGRAM IN PAGE 2:
 95 0022 C180           LD      EREG(P1)        ; IF FIRST LINE LENGTH
 96 0024 E40D           XRI     0D              ; POINTS TO CARR. RETURN
 97 0026 9802           JZ      $0              ; AT END OF LINE
 98 0028 BAF6           DLD     PAGE(P2)        ;IF NOT, PAGE = 1
 99 002A C420   $0:     LDI     020
100 002C 35     $LOOP: XPAH    P1
101 002D C4FF           LDI     -1              ;STORE -1 IN 2 CONSECUTIVE
102 002F C900           ST      (P1)            ; LOCATIONS AT START OF PAGE
103 0031 C901           ST      1(P1)
104 0033 C40D           LDI     0D              ;ALSO PUT A DUMMY END-OF-LINE
105 0035 C9FF           ST      -1(P1)          ; JUST BEFORE TEXT
106 0037 35             XPAH    P1              ;UPDATE P1 TO POINT TO
107 0038 02             CCL                     ; NEXT PAGE (UNTIL PAGE=8)
108 0039 F410           ADI     010             ;REPEAT INITIALIZATION
```

```
 109 003B E480            XRI     080             ; FOR PAGES 2-7
 110 003D 9804            JZ      $1
 111 003F E480            XRI     080
 112 0041 90E9            JMP     $LOOP
 113 0043 C400   $1:      LDI     0               ;CLEAR SOME FLAGS
 114 0045 CAF4            ST      RUNMOD(P2)
 115 0047 CAF5            ST      LISTNG(P2)
 116 0049 C458            LDI     L(BEGIN)        ;INITIALIZE IL PC SO THAT
 117 004B CAFB            ST      PCLOW(P2)       ; NIBL PROGRAM
 118 004D C40C            LDI     H(BEGIN)        ; IS EXECUTED IMMEDIATELY
 119 004F CAFA            ST      PCHIGH(P2)
 120 0051 C400   CLEAR:   LDI     0
 121 0053 CAEA            ST      TEMP(P2)
 122 0055 01              XAE
 123 0056 C400   CLEAR1:  LDI     0               ;SET ALL VARIABLES
 124 0058 CA80            ST      EREG(P2)        ; TO ZERO
 125 005A AAEA            ILD     TEMP(P2)
 126 005C 01              XAE
 127 005D C434            LDI     52
 128 005F 60              XRE
 129 0060 9CF4            JNZ     CLEAR1
 130 0062 C450            LDI     L(AESTK)        ;INITIALIZE SOME STACKS:
 131 0064 CAFD            ST      LSTK(P2)        ; ARITHMETIC STACK,
 132 0066 C47A            LDI     L(DOSTAK)
 133 0068 CAFF            ST      DOPTR(P2)       ; DO/UNTIL STACK,
 134 006A C46A            LDI     L(SBRSTK)
 135 006C CAFC            ST      SBRPTR(P2)      ; GOSUB STACK,
 136 006E C4A6            LDI     L(PCSTAK)
 137 0070 CAF9            ST      PCSTK(P2)       ; I.L. CALL STACK,
 138 0072 C48A            LDI     L(FORSTK)
 139 0074 CAFE            ST      FORPTR(P2)      ; FOR/NEXT STACK
 140
 141
 142              ;****************************************
 143              ;*   INTERMEDIATE LANGUAGE EXECUTOR   *
 144              ;****************************************
 145
 146 0076 C2FB   EXECIL:  LD      PCLOW(P2)       ;SET P3 TO CURRENT
 147 0078 33              XPAL    P3              ; IL PC.
 148 0079 C2FA            LD      PCHIGH(P2)
 149 007B 37              XPAH    P3
 150 007C C701   CHEAT:   LD      @1(P3)
 151 007E 01              XAE                     ;GET NEW I.L. INSTRUCTION
 152 007F C701            LD      @1(P3)          ; INTO P3 THROUGH
 153 0081 33              XPAL    P3              ; OBSCURE METHODS
 154 0082 CAFB            ST      PCLOW(P2)       ;SIMULTANEOUSLY, INCREMENT
 155 0084 40<             LDE                     ; THE I.L. PC BY 2
 156 0085 D40F            ANI     0F              ;REMOVE FLAG FROM INSTRUCTION
 157 0087 DC00            ORI     ./256           ; TURN INTO ACTUAL ADDRESS,
 158 0089 37              XPAH    P3              ; PUT BACK INTO P3
 159 008A CAFA            ST      PCHIGH(P2)
 160 008C 40              LDE
 161 008D D4F0            ANI     0F0             ;CHECK IF I.L. INSTRUCTION
 162 008F E420            XRI     TSTBIT          ; IS A 'TEST'
```

```
163 0091 982F            JZ      TST
164 0093 E4A0            XRI     CALBIT!TSTBIT    ;CHECK FOR I.L. CALL
165 0095 9807            JZ      ILCALL
166 0097 E4C0            XRI     JMPBIT!CALBIT    ;CHECK FOR I.L. JUMP
167 0099 98E1            JZ      CHEAT            ;I.L. JUMP IS TRIVIAL
168 009B 3F      NOJUMP: XPPC    P3               ;MUST BE AN ML SUBROUTINE
169 009C 90D8            JMP     EXECIL           ; IF NONE OF THE ABOVE
170
171
172              ;****************************************
173              ;*      INTERMEDIATE LANGUAGE CALL      *
174              ;****************************************
175
176 009E C2F9    ILCALL: LD      PCSTK(P2)
177 00A0 E4D6            XRI     L(LBUF)          ;CHECK FOR STACK OVERFLOW
178 00A2 9C04            JNZ     ILC1
179 00A4 C40A            LDI     10
180 00A6 9060            JMP     EOA
181 00A8 E4D6    ILC1:   XRI     L(LBUF)          ;RESTORE ACCUMULATOR
182 00AA 33              XPAL    P3               ;SAVE LOW BYTE OF NEW
183 00AB CAEA            ST      TEMP(P2)         ; I.L. PC IN TEMP
184 00AD C410            LDI     H(PCSTAK)        ;POINT P3 AT I.L.
185 00AF 37              XPAH    P3               ; SUBROUTINE STACK
186 00B0 01              XAE                      ;SAVE NEW I.L. PC HIGH IN EX
187 00B1 C2FB            LD      PCLOW(P2)        ;SAVE OLD I.L. PC ON STACK
188 00B3 CF01            ST      @1(P3)
189 00B5 C2FA            LD      PCHIGH(P2)
190 00B7 CF01            ST      @1(P3)
191 00B9 C2EA            LD      TEMP(P2)         ;GET LOW BYTE OF NEW
192 00BB 33              XPAL    P3               ; I.L. PC INTO P3 LOW
193 00BC CAF9            ST      PCSTK(P2)        ;UPDATE I.L. STACK POINTER
194 00BE 40              LDE                      ;GET HIGH BYTE OF NEW
195 00BF 37              XPAH    P3               ; I.L. PC INTO P3 HIGH
196 00C0 90BA    CHEAT1: JMP     CHEAT
197
198
199              ;****************************************
200              ;*       I.L. 'TEST' INSTRUCTION        *
201              ;****************************************
202
203                      .LOCAL
204 00C2 CAE7    TST:    ST      CHRNUM(P2)       ;CLEAR NUMBER OF CHARS SCANNE
205 00C4 C501    $SCAN:  LD      @1(P1)           ;SLEW OFF SPACES
206 00C6 E420            XRI     ' '
207 00C8 98FA            JZ      $SCAN
208 00CA C5FF            LD      @-1(P1)          ;REPOSITION CURSOR
209 00CC C2FA            LD      PCHIGH(P2)       ;POINT P3 AT I.L. TABLE
210 00CE 37              XPAH    P3
211 00CF CAEC            ST      FAILHI(P2)       ;OLD P3 BECOMES THE
212 00D1 C2FB            LD      PCLOW(P2)        ; TEST FAIL ADDRESS
213 00D3 33              XPAL    P3
214 00D4 CAED            ST      FAILLO(P2)
215 00D6 C701    $LOOP:  LD      @1(P3)
216 00D8 01              XAE                      ;SAVE CHAR FROM TABLE
```

```
 217 00D9 BAE7            DLD     CHRNUM(P2)      ;DECREMENT CHAR COUNT
 218 00DB 40              LDE                     ;GET CHAR BACK
 219 00DC D47F            ANI     07F             ;SCRUB OFF FLAG (IF ANY)
 220 00DE E501            XOR     @1(P1)          ;IS CHAR EQUAL TO TEXT CHAR?
 221 00E0 9C07            JNZ     $NEQ            ;NO - END TEST
 222 00E2 40              LDE                     ;YES - BUT IS IT LAST CHAR?
 223 00E3 94F1            JP      $LOOP           ;IF NOT, CONTINUE TO COMPARE
 224 00E5 9095            JMP     CHEAT           ;IF SO, GET NEXT I.L.
 225 00E7 908D   XO:      JMP     EXECIL          ; INSTRUCTION
 226 00E9 C2E7   $NEQ:    LD      CHRNUM(P2)      ;RESTORE P1 TO
 227 00EB 01              XAE                     ; ORIGINAL VALUE
 228 00EC C580            LD      @EREG(P1)
 229 00EE C2ED            LD      FAILLO(P2)      ;LOAD TEST-FAIL ADDRESS
 230 00F0 33              XPAL    P3              ; INTO P3
 231 00F1 C2EC            LD      FAILHI(P2)
 232 00F3 37              XPAH    P3
 233 00F4 90CA            JMP     CHEAT1          ;GET NEXT I.L. INSTRUCTION
 234
 235
 236                      ;*************************************
 237                      ;*        I.L. SUBROUTINE RETURN     *
 238                      ;*************************************
 239
 240 00F6 C410   RTN:     LDI     H(PCSTAK)       ;POINT P3 AT I.L. PC STACK
 241 00F8 37              XPAH    P3
 242 00F9 C2F9            LD      PCSTK(P2)
 243 00FB 33              XPAL    P3
 244 00FC C7FF            LD      @-1(P3)         ;GET HIGH PART OF OLD PC
 245 00FE 01              XAE
 246 00FF C7FF            LD      @-1(P3)         ;GET LOW PART OF OLD PC
 247 0101 33              XPAL    P3
 248 0102 CAF9            ST      PCSTK(P2)       ;UPDATE IL STACK POINTER
 249 0104 40              LDE
 250 0105 37              XPAH    P3              ;P3 NOW HAS OLD IL PC
 251 0106 90B8            JMP     CHEAT1
 252 0108 9041   EOA:     JMP     EO
 253
 254
 255                      ;*************************************
 256                      ;*     SAVE GOSUB RETURN ADDRESS     *
 257                      ;*************************************
 258
 259 010A C2FC   SAV:     LD      SBRPTR(P2)
 260 010C E47A            XRI     L(DOSTAK)       ;CHECK FOR MORE
 261 010E 981C            JZ      SAV2            ; THAN 8 SAVES
 262 0110 AAFC            ILD     SBRPTR(P2)
 263 0112 AAFC            ILD     SBRPTR(P2)
 264 0114 33              XPAL    P3              ;SET P3 TO
 265 0115 C410            LDI     H(SBRSTK)       ; SUBROUTINE STACK TOP.
 266 0117 37              XPAH    P3
 267 0118 C2F4            LD      RUNMOD(P2)      ;IF IMMEDIATE MODE,
 268 011A 980A            JZ      SAV1            ; SAVE NEGATIVE ADDRESS.
 269 011C 35              XPAH    P1              ;SAVE HIGH PORTION
 270 011D CBFF            ST      -1(P3)          ; OF CURSOR
```

```
271 011F 35              XPAH    P1
272 0120 31              XPAL    P1                ;SAVE LOW PORTION
273 0121 CBFE            ST      -2(P3)            ; OF CURSOR
274 0123 31              XPAL    P1
275 0124 90C1            JMP     XO                ;RETURN
276 0126 C4FF   SAV1:    LDI     -1                ;IMMEDIATE MODE
277 0128 CBFF            ST      -1(P3)            ; RETURN ADDRESS IS
278 012A 90BB            JMP     XO                ; NEGATIVE.
279 012C C40A   SAV2:    LDI     10                ;ERROR: MORE THAN
280 012E 901B            JMP     EO                ; 8 GOSUBS
281
282
283             ;****************************************
284             ;*      CHECK STATEMENT FINISHED        *
285             ;****************************************
286
287 0130 C501   DONE:    LD      @1(P1)            ;SKIP SPACES
288 0132 E420            XRI     ' '
289 0134 98FA            JZ      DONE
290 0136 E42D            XRI     ' ' ! OD          ;IS IT CARRIAGE RETURN?
291 0138 9804            JZ      DONE1             ;YES - RETURN
292 013A E437            XRI     037               ;IS CHAR A ':' ?
293 013C 9C01            JNZ     DONE2             ;NO - ERROR
294 013E 3F     DONE1:   XPPC    P3                ;YES - RETURN
295 013F C404   DONE2:   LDI     4
296 0141 9008            JMP     EO
297
298
299             ;****************************************
300             ;*          RETURN FROM GOSUB           *
301             ;****************************************
302
303 0143 C2FC   RSTR:    LD      SBRPTR(P2)
304 0145 E46A            XRI     L(SBRSTK)         ;CHECK FOR RETURN
305 0147 9C04            JNZ     RSTR1             ; W/O GOSUB
306 0149 C409            LDI     9
307 014B 9040   EO:      JMP     E1                ;REPORT THE ERROR
308 014D BAFC   RSTR1:   DLD     SBRPTR(P2)
309 014F BAFC            DLD     SBRPTR(P2)        ;POP GOSUB STACK,
310 0151 33              XPAL    P3                ; PUT PTR INTO P3
311 0152 C410            LDI     H(SBRSTK)
312 0154 37              XPAH    P3
313 0155 C301            LD      1(P3)             ;IF ADDRESS NEGATIVE,
314 0157 9406            JP      RSTR2             ; SUBROUTINE WAS CALLED
315 0159 C400            LDI     0                 ; FROM EDIT MODE,
316 015B CAF4            ST      RUNMOD(P2)        ; SO RETURN TO EDITING
317 015D 9088   X1:      JMP     XO
318 015F 35     RSTR2:   XPAH    P1                ;RESTORE CURSOR HIGH
319 0160 C300            LD      0(P3)
320 0162 31              XPAL    P1                ;RESTORE CURSOR LOW
321 0163 C401            LDI     1                 ;SET RUN MODE
322 0165 CAF4            ST      RUNMOD(P2)
323 0167 90F4            JMP     X1
324
```

```
                        ;*****************************************
                        ;*      TRANSFER TO NEW STATEMENT        *
                        ;*****************************************

0169 C2F2   XFER:   LD      LABLHI(P2)      ;CHECK FOR NON-EXISTENT LINE
016B 9404           JP      XFER1
016D C408           LDI     8
016F 901C           JMP     E1
0171 C401   XFER1:  LDI     1               ;SET RUN MODE TO 1
0173 CAF4           ST      RUNMOD(P2)
0175 3F             XPPC    P3


                        ;*****************************************
                        ;*      PRINT STRING IN TEXT             *
                        ;*****************************************

0176        PRS:    LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
017C C501           LD      @1(P1)          ;LOAD NEXT CHAR
017E E422           XRI     '"'             ;IF ", END OF
0180 98DB           JZ      X1              ; STRING
0182 E42F           XRI     02F             ;IF CR, ERROR
0184 9805           JZ      PRS1
0186 E40D           XRI     0D              ;RESTORE CHAR
0188 3F             XPPC    P3              ;PRINT CHAR
0189 90EB           JMP     PRS             ;GET NEXT CHAR
018B C407   PRS1:   LDI     7               ;SYNTAX ERROR
018D 9035   E1:     JMP     E2


                        ;*****************************************
                        ;*      PRINT NUMBER ON STACK            *
                        ;*****************************************

                        ; THIS ROUTINE IS BASED ON DENNIS ALLISON'S BINARY TO DECIMAL
                        ; CONVERSION ROUTINE IN VOL. 1, #1 OF "DR. DOBB'S JOURNAL",
                        ; BUT IS MUCH MORE OBSCURE BECAUSE OF THE STACK MANIPULATION.

                            .LOCAL
018F C410   PRN:    LDI     H(AESTK)        ;POINT P3 AT A.E. STACK
0191 37             XPAH    P3
0192 AAFD           ILD     LSTK(P2)
0194 AAFD           ILD     LSTK(P2)
0196 33             XPAL    P3
0197 C40A           LDI     10              ;PUT 10 ON STACK (WE'LL BE
0199 CBFE           ST      -2(P3)          ; DIVIDING BY IT LATER)
019B C400           LDI     0
019D CBFF           ST      -1(P3)
019F C405           LDI     5               ;SET CHRNUM TO POINT TO PLACE
01A1 CAE7           ST      CHRNUM(P2)      ; IN STACK WHERE WE STORE
01A3 C4FF           LDI     -1              ; THE CHARACTERS TO PRINT
01A5 CB05           ST      5(P3)           ;FIRST CHAR IS A FLAG (-1)
01A7 C3FD           LD      -3(P3)          ;CHECK IF NUMBER IS NEGATIVE
```

```
 379 01A9 9413           JP      $1
 380 01AB C42D           LDI     '-'             ;PUT '-' ON STACK, AND NEGATE
 381 01AD CB04           ST      4(P3)           ; THE NUMBER
 382 01AF C400           LDI     0
 383 01B1 03             SCL
 384 01B2 FBFC           CAD     -4(P3)
 385 01B4 CBFC           ST      -4(P3)
 386 01B6 C400           LDI     0
 387 01B8 FBFD           CAD     -3(P3)
 388 01BA CBFD           ST      -3(P3)
 389 01BC 909F           JMP     X1              ;GO DO DIVISION BY 10
 390 01BE C420   $1:     LDI     ' '             ;IF POSITIVE, PUT ' ' ON
 391 01C0 CB04           ST      4(P3)           ; STACK BEFORE DIVISION
 392 01C2 9099   X4:     JMP     X1
 393 01C4 9057   E2:     JMP     ERR1
 394
 395             ; THE DIVISION IS PERFORMED, THEN CONTROL IS TRANSFERRED
 396             ; TO PRN1, WHICH FOLLOWS.
 397
 398 01C6 AAFD   PRN1:   ILD     LSTK(P2)        ;POINT P1 AT A.E. STACK
 399 01C8 AAFD           ILD     LSTK(P2)
 400 01CA 31             XPAL    P1
 401 01CB C410           LDI     H(AESTK)
 402 01CD 35             XPAH    P1
 403 01CE AAE7           ILD     CHRNUM(P2)      ;INCREMENT CHARACTER STACK
 404 01D0 01             XAE                     ; POINTER, PUT IN EX. REG.
 405 01D1 C101           LD      1(P1)           ;GET REMAINDER FROM DIVIDE,
 406 01D3 DC30           ORI     '0'
 407 01D5 C980           ST      EREG(P1)        ; PUT IT ON THE STACK
 408 01D7 C1FD           LD      -3(P1)          ;IS THE QUOTIENT ZERO YET?
 409 01D9 D9FC           OR      -4(P1)
 410 01DB 980A           JZ      $PRNT           ;YES - GO PRINT THE NUMBER
 411 01DD C40F           LDI     H(PRNUM1)       ;NO - CHANGE THE I.L. PC
 412 01DF CAFA           ST      PCHIGH(P2)      ; SO THAT DIVIDE IS
 413 01E1 C433           LDI     L(PRNUM1)       ; PERFORMED AGAIN
 414 01E3 CAFB           ST      PCLOW(P2)
 415 01E5 90DB           JMP     X4              ;GO DO DIVISION BY 10 AGAIN
 416 01E7        $PRNT:  LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
 417 01ED C2F5           LD      LISTNG(P2)      ;IF LISTING, SKIP PRINTING
 418 01EF 9C06           JNZ     $2              ; LEADING SPACE
 419 01F1 C104           LD      4(P1)           ;PRINT EITHER '-'
 420 01F3 3F             XPPC    P3              ; OR LEADING SPACE
 421 01F4 C2E7           LD      CHRNUM(P2)      ;GET EX. REG. VALUE BACK
 422 01F6 01             XAE
 423 01F7 C580   $2:     LD      @EREG(P1)       ;POINT P3 AT FIRST CHAR
 424 01F9 C100           LD      (P1)            ; TO BE PRINTED
 425 01FB 3F     $LOOP:  XPPC    P3              ;PRINT THE CHARACTER
 426 01FC C5FF           LD      @-1(P1)         ;GET NEXT CHARACTER
 427 01FE 94FB           JP      $LOOP           ;REPEAT UNTIL = -1
 428 0200 C450           LDI     L(AESTK)
 429 0202 CAFD           ST      LSTK(P2)        ;CLEAR THE A.E. STACK
 430 0204 C2F5           LD      LISTNG(P2)      ;PRINT A TRAILING SPACE
 431 0206 9CBA           JNZ     X4              ; IF NOT LISTING PROGRAM
 432 0208 C420           LDI     ' '
```

```
 433 020A 3F              XPPC    P3
 434 020B 90B5            JMP     X4
 435
 436
 437                ;***************************************
 438                ;*      CARRIAGE RETURN/LINE FEED      *
 439                ;***************************************
 440
 441 020D           NLINE: LDPI    P3,PUTC-1        ;POINT P3 AT PUTC ROUTINE
 442 0213 C40D            LDI     0D               ;CARRIAGE RETURN
 443 0215 3F              XPPC    P3
 444 0216 C40A            LDI     0A               ;LINE FEED
 445 0218 3F              XPPC    P3
 446 0219 90A7    X5:     JMP     X4
 447
 448
 449                ;***************************************
 450                ;*       ERROR  ROUTINE                *
 451                ;***************************************
 452
 453                        .LOCAL
 454 021B C405    ERR:    LDI     5                ;SYNTAX ERROR
 455 021D CAEB    ERR1:   ST      NUM(P2)          ;SAVE ERROR #
 456 021F C2EB    ERR2:   LD      NUM(P2)
 457 0221 CAEA            ST      TEMP(P2)
 458 0223                 LDPI    P3,PUTC-1        ;POINT P3 AT PUTC
 459 0229 C40D            LDI     0D               ;PRINT CR/LF
 460 022B 3F              XPPC    P3
 461 022C C40A            LDI     0A
 462 022E 3F              XPPC    P3
 463 022F                 LDPI    P1,MESGS         ;P1 -> ERROR MESSAGES
 464 0235 BAEB    $1:     DLD     NUM(P2)          ;IS THIS THE RIGHT MESSAGE?
 465 0237 9806            JZ      $MSG             ;YES - GO PRINT IT
 466 0239 C501    $LOOP:  LD      @1(P1)           ;NO - SCAN THROUGH TO
 467 023B 94FC            JP      $LOOP            ;  NEXT MESSAGE
 468 023D 90F6            JMP     $1
 469 023F C501    $MSG:   LD      @1(P1)           ;GET MESSAGE CHAR
 470 0241 3F              XPPC    P3               ;PRINT IT
 471 0242 C1FF            LD      -1(P1)           ;IS MESSAGE DONE?
 472 0244 94F9            JP      $MSG             ;NO - GET NEXT CHAR
 473 0246 C2EA            LD      TEMP(P2)         ;WAS THIS A BREAK MESSAGE?
 474 0248 E40E            XRI     14
 475 024A 980D            JZ      $3               ;YES - SKIP PRINTING 'ERROR'
 476 024C                 LDPI    P1,MESGS         ;NO - PRINT 'ERROR'
 477 0252 C501    $2:     LD      @1(P1)           ;GET CHARACTER
 478 0254 3F              XPPC    P3               ;PRINT IT
 479 0255 C1FF            LD      -1(P1)           ;DONE?
 480 0257 94F9            JP      $2               ;NO - REPEAT LOOP
 481 0259 C2F4    $3:     LD      RUNMOD(P2)       ;DON'T PRINT LINE #
 482 025B 984D            JZ      FIN              ;  IF IMMEDIATE MODE
 483 025D C420            LDI     ' '
 484 025F 3F              XPPC    P3               ;SPACE
 485 0260 C441            LDI     'A'              ;AT
 486 0262 3F              XPPC    P3
```

```
 487 0263 C454            LDI     'T'
 488 0265 3F              XPPC    P3
 489 0266 C410            LDI     H(AESTK)        ;POINT P3 AT A.E.  STACK
 490 0268 37              XPAH    P3
 491 0269 AAFD            ILD     LSTK(P2)
 492 026B AAFD            ILD     LSTK(P2)
 493 026D 33              XPAL    P3
 494 026E C2F7            LD      HILINE(P2)      ;GET HIGH BYTE OF LINE #
 495 0270 CBFF            ST      -1(P3)          ;PUT ON STACK
 496 0272 C2F8            LD      LOLINE(P2)      ;GET LOW BYTE OF LINE #
 497 0274 CBFE            ST      -2(P3)          ;PUT ON STACK
 498 0276 C431            LDI     L(ERRNUM)       ;GO TO PRN
 499 0278 CAFB            ST      PCLOW(P2)
 500 027A C40E            LDI     H(ERRNUM)
 501 027C CAFA            ST      PCHIGH(P2)
 502 027E 9099   X5A:     JMP     X5
 503
 504
 505              ;***************************************
 506              ;*      BREAK, NXT, FIN, & STRT        *
 507              ;***************************************
 508
 509 0280 C40E   BREAK:   LDI      14              ;*** CAUSE A BREAK ***
 510 0282 9099   E3A:     JMP     ERR1
 511                                              ;*** NEXT STATEMENT ***
 512 0284 C2F4   NXT:     LD      RUNMOD(P2)      ;IF IN EDIT MODE,
 513 0286 9822            JZ      FIN             ; STOP EXECUTION
 514 0288 C100            LD      (P1)            ;IF WE HIT END OF FILE,
 515 028A D480            ANI     080             ; FINISH UP THINGS
 516 028C 9C1C            JNZ     FIN
 517 028E 06              CSA                     ;BREAK IF SOMEONE IS
 518 028F D420            ANI     020             ; TYPING ON THE CONSOLE
 519 0291 98ED            JZ      BREAK
 520 0293 C1FF            LD      -1(P1)          ;GET LAST CHARACTER SCANNED
 521 0295 E40D            XRI     0D              ;WAS IT CARRIAGE RETURN?
 522 0297 9C08            JNZ     NXT1            ;YES - SKIP FOLLOWING UPDATES
 523 0299 C501            LD      @1(P1)          ;GET HIGH BYTE OF NEXT LINE #
 524 029B CAF7            ST      HILINE(P2)      ;SAVE IT
 525 029D C502            LD      @2(P1)          ;GET LOW BYTE OF LINE #, SKIP
 526 029F CAF8            ST      LOLINE(P2)      ; LINE LENGTH BYTE
 527 02A1 C40C   NXT1:    LDI     H(STMT)         ;GO TO 'STMT' IN IL TABLE
 528 02A3 CAFA            ST      PCHIGH(P2)
 529 02A5 C486            LDI     L(STMT)
 530 02A7 CAFB            ST      PCLOW(P2)
 531 02A9 3F              XPPC    P3
 532
 533 02AA C400   FIN:     LDI     0               ;*** FINISH EXECUTION ***
 534 02AC CAF4            ST      RUNMOD(P2)      ;CLEAR RUN MODE
 535 02AE C450            LDI     L(AESTK)        ;CLEAR ARITHMETIC STACK
 536 02B0 CAFD            ST      LSTK(P2)
 537 02B2 C41C            LDI     L(START)        ;MODIFY I.L.  PC TO RETURN
 538 02B4 CAFB            ST      PCLOW(P2)       ; TO PROMPT FOR COMMAND
 539 02B6 C40C            LDI     H(START)
 540 02B8 CAFA            ST      PCHIGH(P2)
```

```
 541 02BA C4A6            LDI     L(PCSTAK)
 542 02BC CAF9            ST      PCSTK(P2)
 543 02BE 90BE            JMP     X5A
 544                                              ;*** START EXECUTION ***
 545 02C0 AAF4   STRT:    ILD     RUNMOD(P2)      ;RUN MODE = 1
 546 02C2 C2E9            LD      TEMP2(P2)       ;POINT CURSOR TO
 547 02C4 35              XPAH    P1              ; START OF NIBL PROGRAM
 548 02C5 C2E8            LD      TEMP3(P2)
 549 02C7 31              XPAL    P1
 550 02C8 C46A            LDI     L(SBRSTK)       ;EMPTY SOME STACKS:
 551 02CA CAFC            ST      SBRPTR(P2)      ; GOSUB STACK,
 552 02CC C48A            LDI     L(FORSTK)
 553 02CE CAFE            ST      FORPTR(P2)      ; FOR STACK
 554 02D0 C47A            LDI     L(DOSTAK)
 555 02D2 CAFF            ST      DOPTR(P2)       ; & DO/UNTIL STACK
 556 02D4 3F              XPPC    P3              ;RETURN
 557 02D5 90A7   X6:      JMP     X5A
 558 02D7 90A9   E4:      JMP     E3A
 559
 560
 561                      ;***************************************
 562                      ;*         LIST NIBL PROGRAM           *
 563                      ;***************************************
 564
 565 02D9 C100   LST:     LD      (P1)            ;CHECK FOR END OF FILE
 566 02DB E480            XRI     080
 567 02DD 9418            JP      LST2
 568 02DF C410            LDI     H(AESTK)        ;GET LINE NUMBER ONTO STACK
 569 02E1 37              XPAH    P3
 570 02E2 AAFD            ILD     LSTK(P2)
 571 02E4 AAFD            ILD     LSTK(P2)
 572 02E6 33              XPAL    P3
 573 02E7 C501            LD      @1(P1)
 574 02E9 CBFF            ST      -1(P3)
 575 02EB C501            LD      @1(P1)
 576 02ED CBFE            ST      -2(P3)
 577 02EF C501            LD      @1(P1)          ;SKIP OVER LINE LENGTH'
 578 02F1 C401            LDI     1
 579 02F3 CAF5            ST      LISTNG(P2)      ;SET LISTING FLAG
 580 02F5 90DE            JMP     X6              ;GO PRINT LINE NUMBER
 581 02F7 C400   LST2:    LDI     0
 582 02F9 CAF5            ST      LISTNG(P2)      ;CLEAR LISTING FLAG
 583 02FB C402            JS      P3,NXT          ;GO TO NXT
 584 0302 90D1   X6A:     JMP     X6
 585 0304 90D1   E5:      JMP     E4
 586 0306        LST3:    LDPI    P3,PUTC-1       ;POINT P3 AT PUTC
 587 030C 06     LST4:    CSA
 588 030D D420            ANI     020
 589 030F 98E6            JZ      LST2            ;IF TYPING, STOP
 590 0311 C501            LD      @1(P1)          ;GET NEXT CHAR
 591 0313 E40D            XRI     0D              ;TEST FOR CR
 592 0315 9805            JZ      LST5
 593 0317 E40D            XRI     0D              ;GET CHARACTER
 594 0319 3F              XPPC    P3              ;PRINT CHARACTER
```

```
 595 031A 90F0            JMP     LST4
 596 031C C40D   LST5:    LDI     0D              ;CARRIAGE RETURN
 597 031E 3F              XPPC    P3
 598 031F C40A            LDI     0A              ;LINE FEED
 599 0321 3F              XPPC    P3
 600 0322 02              CCL
 601 0323 C44B            LDI     L(LIST3)
 602 0325 CAFB            ST      PCLOW(P2)
 603 0327 C40C            LDI     H(LIST3)
 604 0329 CAFA            ST      PCHIGH(P2)
 605 032B 90AC            JMP     LST             ;GET NEXT LINE
 606
 607
 608              ;****************************************
 609              ;*           ADD AND SUBTRACT           *
 610              ;****************************************
 611
 612 032D C410    ADD:    LDI     H(AESTK)        ;SET P3 TO CURRENT
 613 032F 37              XPAH    P3              ; STACK LOCATION
 614 0330 BAFD            DLD     LSTK(P2)
 615 0332 BAFD            DLD     LSTK(P2)
 616 0334 33              XPAL    P3
 617 0335 02              CCL
 618 0336 C3FE            LD      -2(P3)          ;REPLACE TWO TOP ITEMS
 619 0338 F300            ADD     0(P3)           ; ON STACK BY THEIR SUM
 620 033A CBFE            ST      -2(P3)
 621 033C C3FF            LD      -1(P3)
 622 033E F301            ADD     1(P3)
 623 0340 CBFF            ST      -1(P3)
 624 0342 90BE    X7:     JMP     X6A
 625
 626 0344 C410    SUB:    LDI     H(AESTK)        ;SET P3 TO CURRENT
 627 0346 37              XPAH    P3              ; STACK LOCATION
 628 0347 BAFD            DLD     LSTK(P2)
 629 0349 BAFD            DLD     LSTK(P2)
 630 034B 33              XPAL    P3
 631 034C 03              SCL
 632 034D C3FE            LD      -2(P3)          ;REPLACE TWO TOP ITEMS
 633 034F FB00            CAD     0(P3)           ; ON STACK BY THEIR
 634 0351 CBFE            ST      -2(P3)          ; DIFFERENCE
 635 0353 C3FF            LD      -1(P3)
 636 0355 FB01            CAD     1(P3)
 637 0357 CBFF            ST      -1(P3)
 638 0359 90A7            JMP     X6A
 639
 640
 641              ;****************************************
 642              ;*           NEGATE                     *
 643              ;****************************************
 644
 645 035B C410    NEG:    LDI     H(AESTK)        ;SET P3 TO CURRENT
 646 035D 37              XPAH    P3              ; STACK LOCATION
 647 035E C2FD            LD      LSTK(P2)
 648 0360 33              XPAL    P3
```

```
 649 0361 03              SCL
 650 0362 C400            LDI     0
 651 0364 FBFE            CAD     -2(P3)          ;NEGATE TOP ITEM ON STACK
 652 0366 CBFE            ST      -2(P3)
 653 0368 C400            LDI     0
 654 036A FBFF            CAD     -1(P3)
 655 036C CBFF            ST      -1(P3)
 656 036E 90D2    X8:     JMP     X7
 657 0370 9092    E6:     JMP     E5
 658
 659
 660              ;****************************************
 661              ;*          MULTIPLY                    *
 662              ;****************************************
 663
 664                      .LOCAL
 665 0372 C410    MUL:    LDI     H(AESTK)        ;SET P3 TO CURRENT
 666 0374 37              XPAH    P3              ; STACK LOCATION
 667 0375 C2FD            LD      LSTK(P2)
 668 0377 33              XPAL    P3              ;DETERMINE SIGN OF PRODUCT,
 669 0378 C3FF            LD      -1(P3)          ; SAVE IN TEMP(P2)
 670 037A E3FD            XOR     -3(P3)
 671 037C CAEA            ST      TEMP(P2)
 672 037E C3FF            LD      -1(P3)          ;CHECK FOR NEGATIVE
 673 0380 940D            JP      $1              ; MULTIPLIER
 674 0382 03              SCL
 675 0383 C400            LDI     0               ;IF NEGATIVE,
 676 0385 FBFE            CAD     -2(P3)          ; NEGATE
 677 0387 CBFE            ST      -2(P3)
 678 0389 C400            LDI     0
 679 038B FBFF            CAD     -1(P3)
 680 038D CBFF            ST      -1(P3)
 681 038F C3FD    $1:     LD      -3(P3)          ;CHECK FOR NEGATIVE
 682 0391 940D            JP      $2              ; MULTIPLICAND
 683 0393 03              SCL
 684 0394 C400            LDI     0               ;IF NEGATIVE,
 685 0396 FBFC            CAD     -4(P3)          ; NEGATE
 686 0398 CBFC            ST      -4(P3)
 687 039A C400            LDI     0
 688 039C FBFD            CAD     -3(P3)
 689 039E CBFD            ST      -3(P3)
 690 03A0 C400    $2:     LDI     0               ;CLEAR WORKSPACE
 691 03A2 CB00            ST      0(P3)
 692 03A4 CB01            ST      1(P3)
 693 03A6 CB02            ST      2(P3)
 694 03A8 CB03            ST      3(P3)
 695 03AA C410            LDI     16              ;SET COUNTER TO 16
 696 03AC CAEB            ST      NUM(P2)
 697 03AE C3FF    $LOOP:  LD      -1(P3)          ;ROTATE MULTIPLIER
 698 03B0 1F              RRL                     ; RIGHT ONE BIT
 699 03B1 CBFF            ST      -1(P3)
 700 03B3 C3FE            LD      -2(P3)
 701 03B5 1F              RRL
 702 03B6 CBFE            ST      -2(P3)
```

```
 703 03B8 06              CSA                     ;CHECK FOR CARRY BIT
 704 03B9 9411            JP      $3              ;IF NOT SET, DON'T DO ADD
 705 03BB 02              CCL
 706 03BC C302            LD      2(P3)           ;ADD MULTIPLICAND
 707 03BE F3FC            ADD     -4(P3)          ;  INTO WORKSPACE
 708 03C0 CB02            ST      2(P3)
 709 03C2 C303            LD      3(P3)
 710 03C4 F3FD            ADD     -3(P3)
 711 03C6 CB03            ST      3(P3)
 712 03C8 9002            JMP     $3
 713 03CA 90A4   E6A:     JMP     E6
 714 03CC 02     $3:      CCL
 715 03CD C303            LD      3(P3)           ;SHIFT WORKSPACE RIGHT BY 1
 716 03CF 1F              RRL
 717 03D0 CB03            ST      3(P3)
 718 03D2 C302            LD      2(P3)
 719 03D4 1F              RRL
 720 03D5 CB02            ST      2(P3)
 721 03D7 C301            LD      1(P3)
 722 03D9 1F               RRL
 723 03DA CB01            ST      1(P3)
 724 03DC C300            LD      0(P3)
 725 03DE 1F              RRL
 726 03DF CB00            ST      0(P3)
 727 03E1 BAEB            DLD     NUM(P2)         ;DECREMENT COUNTER
 728 03E3 9CC9            JNZ     $LOOP           ;LOOP IF NOT ZERO
 729 03E5 9002            JMP     $4
 730 03E7 9085   X9:      JMP     X8
 731 03E9 C2EA   $4:      LD      TEMP(P2)        ;CHECK SIGN WORD
 732 03EB 940D            JP      $EXIT           ;IF BIT7 = 1, NEGATE PRODUCT
 733 03ED 03              SCL
 734 03EE C400            LDI     0
 735 03F0 FB00            CAD     0(P3)
 736 03F2 CB00            ST      0(P3)
 737 03F4 C400            LDI     0
 738 03F6 FB01            CAD     1(P3)
 739 03F8 CB01            ST      1(P3)
 740 03FA C300   $EXIT:   LD      0(P3)           ;PUT PRODUCT ON TOP
 741 03FC CBFC            ST      -4(P3)          ;  OF STACK
 742 03FE C301            LD      1(P3)
 743 0400 CBFD            ST      -3(P3)
 744 0402 BAFD            DLD     LSTK(P2)        ;SUBTRACT 2 FROM
 745 0404 BAFD            DLD     LSTK(P2)        ;  LSTK
 746 0406 90DF            JMP     X9
 747
 748
 749             ;****************************************
 750             ;*              DIVIDE                  *
 751             ;****************************************
 752
 753                     .LOCAL
 754 0408 C410   DIV:     LDI     H(AESTK)
 755 040A 37              XPAH    P3
 756 040B C2FD            LD      LSTK(P2)
```

```
 757 040D 33             XPAL    P3
 758 040E C3FF           LD      -1(P3)          ;CHECK FOR DIVISION BY 0
 759 0410 DBFE           OR      -2(P3)
 760 0412 9C04           JNZ     $0
 761 0414 C40D           LDI     13
 762 0416 90B2           JMP     E6A
 763 0418 C3FD   $0:     LD      -3(P3)
 764 041A E3FF           XOR     -1(P3)
 765 041C CAEA           ST      TEMP(P2)        ;SAVE SIGN OF QUOTIENT
 766 041E C3FD           LD      -3(P3)          ;IS DIVIDEND POSITIVE?
 767 0420 9411           JP      $POS            ;YES - JUMP
 768 0422 C400           LDI     0
 769 0424 03             SCL
 770 0425 FBFC           CAD     -4(P3)          ;NO - NEGATE DIVIDEND,
 771 0427 CB03           ST      3(P3)           ; STORE IN RIGHT HALF
 772 0429 C400           LDI     0               ; OF 32-BIT ACCUMULATOR
 773 042B FBFD           CAD     -3(P3)
 774 042D CB02           ST      2(P3)
 775 042F 900A           JMP     $1
 776 0431 90B4   X9A:    JMP     X9
 777 0433 C3FD   $POS:   LD      -3(P3)          ;STORE NON-NEGATED DIVIDEND
 778 0435 CB02           ST      2(P3)           ; IN 32-BIT ACCUMULATOR
 779 0437 C3FC           LD      -4(P3)
 780 0439 CB03           ST      3(P3)
 781 043B C3FF   $1:     LD      -1(P3)          ;CHECK FOR NEGATIVE DIVISOR
 782 043D 940D           JP      $2
 783 043F C400           LDI     0               ;NEGATE DIVISOR
 784 0441 03             SCL
 785 0442 FBFE           CAD     -2(P3)
 786 0444 CBFE           ST      -2(P3)
 787 0446 C400           LDI     0
 788 0448 FBFF           CAD     -1(P3)
 789 044A CBFF           ST      -1(P3)
 790 044C C400   $2:     LDI     0               ;PUT ZERO IN:
 791 044E CB01           ST      1(P3)           ;  LEFT HALF OF 32-BIT ACC,
 792 0450 CB00           ST      0(P3)
 793 0452 CAEB           ST      NUM(P2)         ;  THE COUNTER, AND
 794 0454 CBFD           ST      -3(P3)          ;  IN THE DIVIDEND, NOW USED
 795 0456 CBFC           ST      -4(P3)          ;  STORE THE QUOTIENT
 796 0458 02     $LOOP:  CCL                     ;BEGIN MAIN DIVIDE LOOP:
 797 0459 C3FC           LD      -4(P3)          ;  SHIFT QUOTIENT LEFT,
 798 045B F3FC           ADD     -4(P3)
 799 045D CBFC           ST      -4(P3)
 800 045F C3FD           LD      -3(P3)
 801 0461 F3FD           ADD     -3(P3)
 802 0463 CBFD           ST      -3(P3)
 803 0465 02             CCL                     ;  SHIFT 32-BIT ACC LEFT,
 804 0466 C303           LD      3(P3)
 805 0468 F303           ADD     3(P3)
 806 046A CB03           ST      3(P3)
 807 046C C302           LD      2(P3)
 808 046E F302           ADD     2(P3)
 809 0470 CB02           ST      2(P3)
 810 0472 C301           LD      1(P3)
```

```
 811 0474 F301            ADD     1(P3)
 812 0476 CB01            ST      1(P3)
 813 0478 C300            LD      (P3)
 814 047A F300            ADD     (P3)
 815 047C CB00            ST      (P3)
 816 047E 03              SCL
 817 047F C301            LD      1(P3)           ;  SUBTRACT DIVISOR INTO
 818 0481 FBFE            CAD     -2(P3)          ;   LEFT HALF OF ACC,
 819 0483 CB01            ST      1(P3)
 820 0485 C300            LD      (P3)
 821 0487 FBFF            CAD     -1(P3)
 822 0489 CB00            ST      (P3)
 823 048B 9411            JP      $ENT1           ;  IF RESULT IS NEGATIVE,
 824 048D 02              CCL                     ;   RESTORE ORIGINAL CONTENTS
 825 048E C301            LD      1(P3)           ;   OF ACC BY ADDING DIVISOR
 826 0490 F3FE            ADD     -2(P3)
 827 0492 CB01            ST      1(P3)
 828 0494 C300            LD      (P3)
 829 0496 F3FF            ADD     -1(P3)
 830 0498 CB00            ST      (P3)
 831 049A 9008            JMP     $3
 832 049C 9093   X9B:     JMP     X9A
 833 049E C3FC   $ENT1:   LD      -4(P3)          ;  ELSE IF RESULT POSITIVE,
 834 04A0 DC01            ORI     1               ;   RECORD A 1 IN QUOTIENT
 835 04A2 CBFC            ST      -4(P3)          ;   W/O RESTORING THE ACC
 836 04A4 AAEB   $3:      ILD     NUM(P2)         ;  INCREMENT THE COUNTER
 837 04A6 E410            XRI     16              ;  ARE WE DONE?
 838 04A8 9CAE            JNZ     $LOOP           ;  LOOP IF NOT DONE
 839 04AA C2EA            LD      TEMP(P2)        ;CHECK THE QUOTIENT'S SIGN,
 840 04AC 940D            JP      $END            ; NEGATING IF NECESSARY
 841 04AE C400            LDI     0
 842 04B0 03              SCL
 843 04B1 FBFC            CAD     -4(P3)
 844 04B3 CBFC            ST      -4(P3)
 845 04B5 C400            LDI     0
 846 04B7 FBFD            CAD     -3(P3)
 847 04B9 CBFD            ST      -3(P3)
 848 04BB BAFD   $END:    DLD     LSTK(P2)        ;DECREMENT THE STACK POINTER,
 849 04BD BAFD            DLD     LSTK(P2)
 850 04BF 90DB            JMP     X9B             ;  AND EXIT
 851
 852
 853                      ;****************************************
 854                      ;*          STORE VARIABLE              *
 855                      ;****************************************
 856
 857 04C1 C410   STORE:   LDI     H(AESTK)        ;SET P3 TO STACK
 858 04C3 37              XPAH    P3
 859 04C4 C2FD            LD      LSTK(P2)
 860 04C6 33              XPAL    P3
 861 04C7 C7FD            LD      @-3(P3)         ;GET VARIABLE INDEX
 862 04C9 01              XAE                     ;PUT IN E REG
 863 04CA C301            LD      1(P3)
 864 04CC CA80            ST      EREG(P2)        ;STORE LOWER 8 BITS
```

```
 865 04CE 02              CCL                     ; INTO VARIABLE
 866 04CF 40              LDE                     ;INCREMENT INDEX
 867 04D0 F401            ADI     1
 868 04D2 01              XAE
 869 04D3 C302            LD      2(P3)
 870 04D5 CA80            ST      EREG(P2)        ;STORE UPPER 8 BITS
 871 04D7 33              XPAL    P3              ; INTO VARIABLE
 872 04D8 CAFD            ST      LSTK(P2)        ;UPDATE STACK POINTER
 873 04DA C400    X10:    JS      P3,EXECIL
 874
 875
 876              ;****************************************
 877              ;*      TEST FOR VARIABLE IN TEXT       *
 878              ;****************************************
 879
 880 04E1 C501    TSTVAR: LD      @1(P1)
 881 04E3 E420            XRI     ' '             ;SLEW OFF SPACES
 882 04E5 98FA            JZ      TSTVAR
 883 04E7 C1FF            LD      -1(P1)          ;GET CHARACTER IN QUESTION
 884 04E9 03              SCL
 885 04EA FC5B            CAI     'Z'+1           ;SUBTRACT 'Z'+1
 886 04EC 9405            JP      $FAIL           ;NOT VARIABLE IF POSITIVE
 887 04EE 03              SCL
 888 04EF FCE6            CAI     'A'-'Z'-1       ;SUBTRACT 'A'
 889 04F1 9412            JP      $MAYBE          ;IF POS, MAY BE VARIABLE
 890 04F3 C5FF    $FAIL:  LD      @-1(P1)         ;BACKSPACE CURSOR
 891 04F5 C2FB            LD      PCLOW(P2)       ;GET TEST-FAIL ADDRESS
 892 04F7 33              XPAL    P3              ; FROM I.L. TABLE, PUT IT
 893 04F8 C2FA            LD      PCHIGH(P2)      ; INTO I.L. PROGRAM COUNTER
 894 04FA 37              XPAH    P3
 895 04FB C300            LD      (P3)
 896 04FD CAFA            ST      PCHIGH(P2)
 897 04FF C301            LD      1(P3)
 898 0501 CAFB            ST      PCLOW(P2)
 899 0503 90D5            JMP     X10
 900 0505 01      $MAYBE: XAE                     ;SAVE VALUE (0-25)
 901 0506 C100            LD      (P1)            ;CHECK FOLLOWING CHAR
 902 0508 03              SCL                     ;MUST NOT BE A LETTER
 903 0509 FC5B            CAI     'Z'+1           ; OTHERWISE WE'D BE LOOKING
 904 050B 9405            JP      $OK             ; AT A KEYWORD, NOT VARIABLE
 905 050D 03              SCL
 906 050E FCE6            CAI     'A'-'Z'-1
 907 0510 94E1            JP      $FAIL
 908 0512 C410    $OK:    LDI     H(AESTK)        ;SET P3 TO CURRENT
 909 0514 37              XPAH    P3              ; STACK LOCATION
 910 0515 AAFD            ILD     LSTK(P2)        ;INCR STACK POINTER
 911 0517 33              XPAL    P3
 912 0518 02              CCL                     ;DOUBLE VARIABLE INDEX
 913 0519 40              LDE
 914 051A 70              ADE
 915 051B CBFF            ST      -1(P3)          ;PUT INDEX ON STACK
 916 051D C402            LDI     2               ;INCREMENT I.L. PC, SKIPPING
 917 051F 02              CCL                     ; OVER TEST-FAIL ADDRESS
 918 0520 F2FB            ADD     PCLOW(P2)
```

```
 919 0522 CAFB            ST      PCLOW(P2)
 920 0524 C400            LDI     0
 921 0526 F2FA            ADD     PCHIGH(P2)
 922 0528 CAFA            ST      PCHIGH(P2)
 923 052A 90AE            JMP     X10
 924
 925
 926              ;*************************************
 927              ;*    IND -- EVALUATE A VARIABLE     *
 928              ;*************************************
 929
 930 052C C410    IND:    LDI     H(AESTK)        ;SET P3 TO STACK
 931 052E 37              XPAH    P3
 932 052F AAFD            ILD     LSTK(P2)
 933 0531 33              XPAL    P3
 934 0532 C3FE            LD      -2(P3)          ;GET INDEX OFF TOP
 935 0534 01              XAE                     ;PUT INDEX IN E REG
 936 0535 C280            LD      EREG(P2)        ;GET LOWER 8 BITS
 937 0537 CBFE            ST      -2(P3)          ;SAVE ON STACK
 938 0539 02              CCL
 939 053A 40              LDE                     ;INCREMENT E REG
 940 053B F401            ADI     1
 941 053D 01              XAE
 942 053E C280            LD      EREG(P2)        ;GET UPPER 8 BITS
 943 0540 CBFF            ST      -1(P3)          ;SAVE ON STACK
 944 0542 9096    X11:    JMP     X10
 945
 946
 947              ;*************************************
 948              ;*      RELATIONAL OPERATORS         *
 949              ;*************************************
 950
 951 0544 C401    EQ:     LDI     1               ;EACH RELATIONAL OPERATOR
 952 0546 9012            JMP     CMP             ; LOADS A NUMBER USED LATER
 953 0548 C402    NEQ:    LDI     2               ; AS A CASE SELECTOR, AFTER
 954 054A 900E            JMP     CMP             ; THE TWO OPERANDS ARE COM-
 955 054C C403    LSS:    LDI     3               ; PARED.  BASED ON THE COM-
 956 054E 900A            JMP     CMP             ; PARISON, FLAGS ARE SET THAT
 957 0550 C404    LEQ:    LDI     4               ; ARE EQUIVALENT TO THOSE SET
 958 0552 9006            JMP     CMP             ; BY THE 'CMP' INSTRUCTION IN
 959 0554 C405    GTR:    LDI     5               ; THE PDP-11.  THESE PSEUDO-
 960 0556 9002            JMP     CMP             ; FLAGS ARE USED TO DETERMINE
 961 0558 C406    GEQ:    LDI     6               ; WHETHER THE PARTICULAR
 962                                              ; RELATION IS SATISFIED OR NO
 963 055A CAEB    CMP:    ST      NUM(P2)
 964 055C C410            LDI     H(AESTK)        ;SET P3 -> ARITH STACK
 965 055E 37              XPAH    P3
 966 055F BAFD            DLD     LSTK(P2)
 967 0561 BAFD            DLD     LSTK(P2)
 968 0563 33              XPAL    P3
 969 0564 03              SCL
 970 0565 C3FE            LD      -2(P3)          ;SUBTRACT THE TWO OPERANDS,
 971 0567 FB00            CAD     (P3)            ; STORING RESULT IN LO & HI
 972 0569 CAEF            ST      LO(P2)
```

```
 973 056B C3FF            LD      -1(P3)
 974 056D FB01            CAD     1(P3)
 975 056F CAEE            ST      HI(P2)
 976 0571 E3FF            XOR     -1(P3)          ;OVERFLOW OCCURS IF SIGNS OF
 977 0573 01              XAE                     ;  RESULT AND 1ST OPERAND
 978 0574 C3FF            LD      -1(P3)          ;  DIFFER, AND SIGNS OF THE
 979 0576 E301            XOR     1(P3)           ;  TWO OPERANDS DIFFER
 980 0578 50              ANE                     ;BIT 7 EQUIVALENT TO V FLAG
 981 0579 E2EE            XOR     HI(P2)          ;BIT 7 EQUIVALENT TO N XOR V
 982 057B CAEA            ST      TEMP(P2)        ;STORE IN TEMP
 983 057D C2EE            LD      HI(P2)          ;DETERMINE IF RESULT WAS ZERO
 984 057F DAEF            OR      LO(P2)
 985 0581 9802            JZ      SETZ            ;IF RESULT=0, SET Z FLAG
 986 0583 C480            LDI     080             ;  ELSE CLEAR Z FLAG
 987 0585 E480   SETZ:    XRI     080
 988 0587 01              XAE                     ;BIT 7 OF EX = Z FLAG
 989
 990 0588 BAEB            DLD     NUM(P2)         ;TEST FOR =
 991 058A 9C05            JNZ     NEQ1
 992 058C 40              LDE                     ;  EQUAL IF Z = 1
 993 058D 902B            JMP     CMP1
 994 058F 90B1   X12:     JMP     X11
 995 0591 BAEB   NEQ1:    DLD     NUM(P2)         ;TEST FOR <>
 996 0593 9C05            JNZ     LSS1
 997 0595 40              LDE                     ;  NOT EQUAL IF Z = 0
 998 0596 E480            XRI     080
 999 0598 9020            JMP     CMP1
1000 059A BAEB   LSS1:    DLD     NUM(P2)         ;TEST FOR <
1001 059C 9C04            JNZ     LEQ1
1002 059E C2EA            LD      TEMP(P2)        ;  LESS THAN IF (N XOR V)=1
1003 05A0 9018            JMP     CMP1
1004 05A2 BAEB   LEQ1:    DLD     NUM(P2)         ;TEST FOR <=
1005 05A4 9C05            JNZ     GTR1
1006 05A6 40              LDE                     ;  LESS THAN OR EQUAL
1007 05A7 DAEA            OR      TEMP(P2)        ;    IF (Z OR (N XOR V))=1
1008 05A9 900F            JMP     CMP1
1009 05AB BAEB   GTR1:    DLD     NUM(P2)         ;TEST FOR >
1010 05AD 9C07            JNZ     GEQ1
1011 05AF 40              LDE                     ;  GREATER THAN
1012 05B0 DAEA            OR      TEMP(P2)        ;    IF (Z OR (N XOR V))=0
1013 05B2 E480            XRI     080
1014 05B4 9004            JMP     CMP1
1015 05B6 C2EA   GEQ1:    LD      TEMP(P2)        ;GREATER THAN OR EQUAL
1016 05B8 E480            XRI     080             ;  IF (N XOR V)=0
1017 05BA 9404   CMP1:    JP      FALSE           ;IS RELATION SATISFIED?
1018 05BC C401            LDI     1               ;YES - PUSH 1 ON STACK
1019 05BE 9002            JMP     CMP2
1020 05C0 C400   FALSE:   LDI     0               ;NO - PUSH 0 ON STACK
1021 05C2 CBFE   CMP2:    ST      -2(P3)
1022 05C4 C400            LDI     0
1023 05C6 CBFF            ST      -1(P3)
1024 05C8 C400            JS      P3,RTN          ;DO AN I.L. RETURN
1025 05CF 90BE            JMP     X12
1026
```

```
1027
1028                ;****************************************
1029                ;*    IF STATEMENT TEST FOR ZERO        *
1030                ;****************************************
1031
1032 05D1 C2EF   CMPR:   LD      LO(P2)          ;GET LOW & HI BYTES OF EXPR.
1033 05D3 DAEE           OR      HI(P2)          ;TEST IF EXPRESSION IS ZERO
1034 05D5 9802           JZ      FAIL            ;YES - IT IS
1035 05D7 90B6           JMP     X12             ;NO - IT ISN'T SO CONTINUE
1036 05D9 C501   FAIL:   LD      @1(P1)          ;SKIP TO NEXT LINE IN PROGRAM
1037 05DB E40D           XRI     0D              ; (I.E. TIL NEXT CR)
1038 05DD 9CFA           JNZ     FAIL
1039 05DF C402           JS      P3,NXT          ;CALL NXT AND RETURN
1040 05E6 90A7   X12A:   JMP     X12
1041
1042
1043                ;****************************************
1044                ;*         AND, OR, & NOT               *
1045                ;****************************************
1046
1047                    .LOCAL
1048 05E8 C401   ANDOP:  LDI     1               ;EACH OPERATION HAS ITS
1049 05EA 9006           JMP     $1              ; OWN CASE SELECTOR.
1050 05EC C402   OROP:   LDI     2
1051 05EE 9002           JMP     $1
1052 05F0 C403   NOTOP:  LDI     3
1053 05F2 CAEB   $1:     ST      NUM(P2)
1054 05F4 C410           LDI     H(AESTK)        ;SET P3 -> ARITH. STACK
1055 05F6 37             XPAH    P3
1056 05F7 BAFD           DLD     LSTK(P2)
1057 05F9 BAFD           DLD     LSTK(P2)
1058 05FB 33             XPAL    P3
1059 05FC BAEB           DLD     NUM(P2)         ;TEST FOR 'AND'
1060 05FE 9C0E           JNZ     $OR
1061 0600 C301           LD      1(P3)           ;REPLACE TWO TOP ITEMS ON
1062 0602 D3FF           AND     -1(P3)          ; STACK BY THEIR 'AND'
1063 0604 CBFF           ST      -1(P3)
1064 0606 C300           LD      0(P3)
1065 0608 D3FE           AND     -2(P3)
1066 060A CBFE           ST      -2(P3)
1067 060C 90D8           JMP     X12A
1068 060E BAEB   $OR:    DLD     NUM(P2)         ;TEST FOR 'OR'
1069 0610 9C0E           JNZ     $NOT
1070 0612 C301           LD      1(P3)           ;REPLACE TWO TOP ITEMS ON
1071 0614 DBFF           OR      -1(P3)          ; STACK BY THEIR 'OR'
1072 0616 CBFF           ST      -1(P3)
1073 0618 C300           LD      0(P3)
1074 061A DBFE           OR      -2(P3)
1075 061C CBFE           ST      -2(P3)
1076 061E 90C6           JMP     X12A
1077 0620 C701   $NOT:   LD      @1(P3)          ;'NOT' OPERATION
1078 0622 E4FF           XRI     0FF
1079 0624 CBFF           ST      -1(P3)          ;REPLACE TOP ITEM ON STACK
1080 0626 C701           LD      @1(P3)          ; BY ITS ONE'S COMPLEMENT
```

```
1081 0628 E4FF            XRI     OFF
1082 062A CBFF            ST      -1(P3)
1083 062C 33              XPAL    P3
1084 062D CAFD            ST      LSTK(P2)          ;STACK POINTER FIXUP
1085 062F 90B5   X12B:    JMP     X12A
1086
1087
1088              ;***************************************
1089              ;*     EXCHANGE CURSOR WITH RAM        *
1090              ;***************************************
1091
1092 0631 C2F1   XCHGP1:  LD      P1LOW(P2)         ;THIS ROUTINE IS HANDY WHEN
1093 0633 31              XPAL    P1                ; EXECUTING AN 'INPUT' STMT
1094 0634 CAF1            ST      P1LOW(P2)         ; IT EXCHANGES THE CURRENT
1095 0636 C2F0            LD      P1HIGH(P2)        ; TEXT CURSOR WITH ONE SAVED
1096 0638 35              XPAH    P1                ; IN RAM
1097 0639 CAF0            ST      P1HIGH(P2)
1098 063B 3F              XPPC    P3
1099
1100
1101              ;***************************************
1102              ;*          CHECK RUN MODE             *
1103              ;***************************************
1104
1105 063C C2F4   CKMODE:  LD      RUNMOD(P2)        ;THIS ROUTINE CAUSES AN ERROR
1106 063E 9801            JZ      CK1               ; IF CURRENTLY IN EDIT MODE
1107 0640 3F              XPPC    P3
1108 0641 C403   CK1:     LDI     3
1109 0643 CAEB   E8:      ST      NUM(P2)           ;ERROR IF RUN MODE = 0
1110 0645 C402            JS      P3,ERR2           ;MINOR KLUGE
1111
1112
1113              ;***************************************
1114              ;*       GET HEXADECIMAL NUMBER        *
1115              ;***************************************
1116
1117                      .LOCAL
1118 064C AAFD   HEX:     ILD     LSTK(P2)          ;POINT P3 AT ARITH STACK
1119 064E AAFD            ILD     LSTK(P2)
1120 0650 33              XPAL    P3
1121 0651 C410            LDI     H(AESTK)
1122 0653 37              XPAH    P3
1123 0654 C400            LDI     0                 ;NUMBER INITIALLY ZERO
1124 0656 CBFF            ST      -1(P3)            ;PUT IT ON STACK
1125 0658 CBFE            ST      -2(P3)
1126 065A CAEB            ST      NUM(P2)           ;ZERO NUMBER OF DIGITS
1127 065C C501   $SKIP:   LD      @1(P1)            ;SKIP ANY SPACES
1128 065E E420            XRI     ' '
1129 0660 98FA            JZ      $SKIP
1130 0662 C5FF            LD      @-1(P1)
1131 0664 C100   $LOOP:   LD      (P1)              ;GET A CHARACTER
1132 0666 03              SCL
1133 0667 FC3A            CAI     '9'+1             ;CHECK FOR A NUMERIC CHAR
1134 0669 9409            JP      $LETR
```

```
1135 066B 03              SCL
1136 066C FCF6            CAI     '0'-'9'-1         ;IF NUMERIC, SHIFT NUMBER
1137 066E 9413            JP      $ENTER            ; AND ADD NEW HEX DIGIT
1138 0670 9032            JMP     $END
1139 0672 90BB   X12C:    JMP     X12B
1140 0674 03     $LETR:   SCL                       ;CHECK FOR HEX LETTER
1141 0675 FC0D            CAI     'G'-'9'-1
1142 0677 942B            JP      $END
1143 0679 03              SCL
1144 067A FCFA            CAI     'A'-'G'
1145 067C 9402            JP      $OK
1146 067E 9024            JMP     $END
1147 0680 02     $OK:     CCL                       ;ADD 10 TO GET TRUE VALUE
1148 0681 F40A            ADI     10                ; OF LETTER
1149 0683 01     $ENTER:  XAE                       ;NEW DIGIT IN EX REG
1150 0684 C404            LDI     4                 ;SET SHIFT COUNTER
1151 0686 CAEA            ST      TEMP(P2)
1152 0688 CAEB            ST      NUM(P2)           ;DIGIT COUNT IS NON-ZERO
1153 068A C3FE   $SHIFT:  LD      -2(P3)            ;SHIFT NUMBER LEFT BY 4
1154 068C 02              CCL
1155 068D F3FE            ADD     -2(P3)
1156 068F CBFE            ST      -2(P3)
1157 0691 C3FF            LD      -1(P3)
1158 0693 F3FF            ADD     -1(P3)
 159 0695 CBFF            ST      -1(P3)
1160 0697 BAEA            DLD     TEMP(P2)
1161 0699 9CEF            JNZ     $SHIFT
1162 069B C3FE            LD      -2(P3)            ;ADD NEW DIGIT
1163 069D 58              ORE                       ; INTO NUMBER
1164 069E CBFE            ST      -2(P3)
1165 06A0 C501            LD      @1(P1)            ;ADVANCE THE CURSOR
1166 06A2 90C0            JMP     $LOOP             ;GET NEXT CHAR
1167 06A4 C2EB   $END:    LD      NUM(P2)           ;CHECK IF THERE WERE
1168 06A6 9C87            JNZ     X12B              ; MORE THAN 0 CHARACTERS
1169 06A8 C405            LDI     5                 ;ERROR IF THERE WERE NONE
1170 06AA 9097   E8B:     JMP     E8
1171
1172
1173              ;****************************************
1174              ;*       TEST FOR NUMBER IN TEXT        *
1175              ;****************************************
1176
1177               ; THIS ROUTINE TESTS FOR A NUMBER IN THE TEXT.  IF NO
1178              ; NUMBER IS FOUND, I.L. CONTROL PASSES TO THE ADDRESS
1179              ; INDICATED IN THE 'TSTN' INSTRUCTION.  OTHERWISE, THE
1180              ; NUMBER IS SCANNED AND PUT ON THE ARITHMETIC STACK,
1181              ; WITH I.L. CONTROL PASSING TO THE NEXT INSTRUCTION.
1182
1183                        .LOCAL
1184 06AC C501   TSTNUM:   LD      @1(P1)
 185 06AE E420             XRI     ' '               ;SKIP OVER ANY SPACES
 186 06B0 98FA             JZ      TSTNUM
1187 06B2 C5FF            LD      @-1(P1)           ;GET FIRST CHAR
1188 06B4 03              SCL                       ;TEST FOR DIGIT
```

```
1189 06B5 FC3A           CAI     '9'+1
1190 06B7 9405           JP      $ABORT
1191 06B9 03             SCL
1192 06BA FCF6           CAI     '0'-'9'-1
1193 06BC 9421           JP      $1
1194 06BE C2FB  $ABORT:  LD      PCLOW(P2)      ;GET TEST-FAIL ADDRESS
1195 06C0 33             XPAL    P3             ; FROM I.L. TABLE
1196 06C1 C2FA           LD      PCHIGH(P2)
1197 06C3 37             XPAH    P3
1198 06C4 C300           LD      (P3)           ;PUT TEST-FAIL ADDRESS
1199 06C6 CAFA           ST      PCHIGH(P2)     ; INTO I.L. PC
1200 06C8 C301           LD      1(P3)
1201 06CA CAFB           ST      PCLOW(P2)
1202 06CC 90A4           JMP     X12C
1203 06CE C402  $RET:    LDI     2              ;SKIP OVER ONE IL INSTRUCTION
1204 06D0 02             CCL                    ; IF NUMBER IS DONE
1205 06D1 F2FB           ADD     PCLOW(P2)
1206 06D3 CAFB           ST      PCLOW(P2)
1207 06D5 C400           LDI     0
1208 06D7 F2FA           ADD     PCHIGH(P2)
1209 06D9 CAFA           ST      PCHIGH(P2)
1210 06DB 9095  X13:     JMP     X12C
1211 06DD 90CB  E8A:     JMP     E8B
1212 06DF 01    $1:      XAE                    ;SAVE DIGIT IN EX REG
1213 06E0 C410           LDI     H(AESTK)       ;POINT P3 AT AE STACK
1214 06E2 37             XPAH    P3
1215 06E3 AAFD           ILD     LSTK(P2)
1216 06E5 AAFD           ILD     LSTK(P2)
1217 06E7 33             XPAL    P3
1218 06E8 C400           LDI     0
1219 06EA CBFF           ST      -1(P3)
1220 06EC 40             LDE
1221 06ED CBFE           ST      -2(P3)
1222 06EF C501  $LOOP:   LD      @1(P1)         ;GET NEXT CHAR
1223 06F1 C100           LD      (P1)
1224 06F3 03             SCL                    ;TEST IF IT IS DIGIT
1225 06F4 FC3A           CAI     '9'+1
1226 06F6 94D6           JP      $RET           ;RETURN IF IT ISN'T
1227 06F8 03             SCL
1228 06F9 FCF6           CAI     '0'-'9'-1
1229 06FB 9402           JP      $2
1230 06FD 90CF           JMP     $RET
1231 06FF 01    $2:      XAE                    ;SAVE DIGIT
1232 0700 C3FF           LD      -1(P3)         ;PUT RESULT IN SCRATCH SPACE
1233 0702 CB01           ST      1(P3)
1234 0704 C3FE           LD      -2(P3)
1235 0706 CB00           ST      (P3)
1236 0708 C402           LDI     2
1237 070A CAEA           ST      TEMP(P2)       ;MULTIPLY RESULT BY 10
1238 070C 02    $SHIFT:  CCL                    ;FIRST MULTIPLY BY 4
1239 070D C3FE           LD      -2(P3)
1240 070F F3FE           ADD     -2(P3)
1241 0711 CBFE           ST      -2(P3)
1242 0713 C3FF           LD      -1(P3)
```

```
1243 0715 F3FF            ADD     -1(P3)
1244 0717 CBFF            ST      -1(P3)
1245 0719 D480            ANI     080             ;MAKE SURE NO OVERFLOW
1246 071B 9C34            JNZ     $ERR            ; OCCURRED
1247 071D BAEA            DLD     TEMP(P2)
1248 071F 9CEB            JNZ     $SHIFT
1249 0721 02              CCL                     ;THEN ADD OLD RESULT,
1250 0722 C3FE            LD      -2(P3)          ; SO WE HAVE RESULT * 5
1251 0724 F300            ADD     (P3)
1252 0726 CBFE            ST      -2(P3)
1253 0728 C3FF            LD      -1(P3)
1254 072A F301            ADD     1(P3)
1255 072C CBFF            ST      -1(P3)
1256 072E D480            ANI     080             ;MAKE SURE NO OVERFLOW
1257 0730 9C1F            JNZ     $ERR            ; OCCURRED
1258 0732 02              CCL                     ;THEN MULTIPLY BY TWO
1259 0733 C3FE            LD      -2(P3)
1260 0735 F3FE            ADD     -2(P3)
1261 0737 CBFE            ST      -2(P3)
1262 0739 C3FF            LD      -1(P3)
1263 073B F3FF            ADD     -1(P3)
1264 073D CBFF            ST      -1(P3)
1265 073F D480            ANI     080             ;MAKE SURE NO OVERFLOW
1266 0741 9C0E            JNZ     $ERR            ; OCCURRED
1267 0743 02              CCL                     ;THEN ADD IN NEW DIGIT
1268 0744 40              LDE
1269 0745 F3FE            ADD     -2(P3)
1270 0747 CBFE            ST      -2(P3)
1271 0749 C400            LDI     0
1272 074B F3FF            ADD     -1(P3)
1273 074D CBFF            ST      -1(P3)
1274 074F 949E            JP      $LOOP           ;REPEAT IF NO OVERFLOW
1275 0751 C406  $ERR:     LDI     6
1276 0753 9088  E9:       JMP     E8A             ;ELSE REPORT ERROR
1277 0755 9084  X14:      JMP     X13
1278
1279
1280            ;****************************************
1281            ;*     GET LINE FROM TELETYPE           *
1282            ;****************************************
1283
1284                      .LOCAL
1285 0757        GETL:    LDPI    P1,LBUF         ;SET P1 TO LBUF
1286 075D C400            LDI     0               ;CLEAR NO.  OF CHAR
1287 075F CAE7            ST      CHRNUM(P2)
1288 0761                 LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
1289 0767 C2F4            LD      RUNMOD(P2)      ;PRINT '? ' IF RUNNING
1290 0769 9808            JZ      $0              ; (I.E.  DURING 'INPUT')
1291 076B C43F            LDI     '?'
1292 076D 3F              XPPC    P3
1293 076E C420            LDI     ' '
1294 0770 3F              XPPC    P3
1295 0771 9003            JMP     $1
1296 0773 C43E  $0:       LDI     '>'             ;OTHERWISE PRINT '>'
```

```
1297 0775 3F             XPPC    P3
1298 0776 C40F   $1:     JS      P3,GECO         ;GET CHARACTER
1299 077D C4C1           LDI     L(PUTC)-1       ;POINT P3 AT PUTC AGAIN
1300 077F 33             XPAL    P3
1301 0780 40             LDE                     ;GET TYPED CHAR
1302 0781 98F3           JZ      $1              ;IGNORE NULLS
1303 0783 E40A           XRI     0A              ;IGNORE LINE FEED
1304 0785 98EF           JZ      $1
1305 0787 40             LDE
1306 0788 E40D           XRI     0D              ;CHECK FOR CR
1307 078A 9850           JZ      $CR
1308 078C 40             LDE
1309 078D E45F           XRI     'O'+010         ;CHECK FOR SHIFT/O
1310 078F 9841           JZ      $RUB
1311 0791 40             LDE                     ;CHECK FOR CTRL/H
1312 0792 E408           XRI     8
1313 0794 9836           JZ      $XH
1314 0796 40             LDE
1315 0797 E415           XRI     015             ;CHECK FOR CTRL/U
1316 0799 980F           JZ      $XU
1317 079B 40             LDE
1318 079C E403           XRI     3               ;CHECK FOR CTRL/C
1319 079E 9C1A           JNZ     $ENTER
1320 07A0 C45E           LDI     '^'             ;ECHO CONTROL/C AS ^C
1321 07A2 3F             XPPC    P3
1322 07A3 C443           LDI     'C'
1323 07A5 3F             XPPC    P3
1324 07A6 C40E           LDI     14              ;CAUSE A BREAK
1325 07A8 90A9           JMP     E9
1326 07AA C45E   $XU:    LDI     '^'             ;ECHO CONTROL/U AS ^U
1327 07AC 3F             XPPC    P3
1328 07AD C455           LDI     'U'
1329 07AF 3F             XPPC    P3
1330 07B0 C40D           LDI     0D              ;PRINT CR/LF
1331 07B2 3F             XPPC    P3
1332 07B3 C40A           LDI     0A
1333 07B5 3F             XPPC    P3
1334 07B6 909F   $2:     JMP     GETL            ;GO GET ANOTHER LINE
1335 07B8 909B   X15:    JMP     X14
1336 07BA 40     $ENTER: LDE
1337 07BB CD01           ST      @1(P1)          ;PUT CHAR IN LBUF
1338 07BD AAE7           ILD     CHRNUM(P2)      ;INCREMENT CHRNUM
1339 07BF E448           XRI     72              ;IF=72, LINE FULL
1340 07C1 9CB3           JNZ     $1
1341 07C3 C40D           LDI     0D
1342 07C5 01             XAE                     ;SAVE CARRIAGE RET
1343 07C6 40             LDE
1344 07C7 3F             XPPC    P3              ;PRINT IT
1345 07C8 9012           JMP     $CR             ;STORE IT IN LBUF
1346 07CA 9087   E10:    JMP     E9
1347 07CC C420   $XH:    LDI     ' '             ;BLANK OUT THE CHARACTER
1348 07CE 3F             XPPC    P3
1349 07CF C408           LDI     8               ;PRINT ANOTHER BACKSPACE
1350 07D1 3F             XPPC    P3
```

```
 351 07D2 C2E7   $RUB:   LD      CHRNUM(P2)
1352 07D4 98A0          JZ      $1
1353 07D6 BAE7          DLD     CHRNUM(P2)      ;ONE LESS CHAR
1354 07D8 C5FF          LD      @-1(P1)         ;BACKSPACE CURSOR
1355 07DA 909A          JMP     $1
1356 07DC 40    $CR:    LDE
1357 07DD CD01          ST      @1(P1)          ;STORE CR IN LBUF
1358 07DF C40A          LDI     0A              ;PRINT LINE FEED
1359 07E1 3F            XPPC    P3
1360 07E2 C410          LDI     H(LBUF)         ;SET P1 TO BEGIN-
1361 07E4 35            XPAH    P1              ; NING OF LBUF
1362 07E5 C4D6          LDI     L(LBUF)
1363 07E7 31            XPAL    P1
1364 07E8 90CE  X16:    JMP     X15
1365
1366
1367            ;***************************************
1368            ;*      EVAL -- GET MEMORY CONTENTS    *
1369            ;***************************************
1370
1371            ; THIS ROUTINE IMPLEMENTS THE '@' OPERATOR IN EXPRESSIONS
1372
1373 07EA C410 EVAL:   LDI     H(AESTK)
1374 07EC 37            XPAH    P3
1375 07ED C2FD          LD      LSTK(P2)
1376 07EF 33            XPAL    P3              ;P3 -> ARITH STACK
 377 07F0 C3FF          LD      -1(P3)          ;GET ADDR OFF STACK,
1378 07F2 35            XPAH    P1              ; AND INTO P1,
1379 07F3 01            XAE                     ; SAVING OLD P1 IN EX & LO
1380 07F4 C3FE          LD      -2(P3)
1381 07F6 31            XPAL    P1
1382 07F7 CAEF          ST      LO(P2)
1383 07F9 C100          LD      0(P1)           ;GET MEMORY CONTENTS,
1384 07FB CBFE          ST      -2(P3)          ; SHOVE ONTO STACK
1385 07FD C400          LDI     0
1386 07FF CBFF          ST      -1(P3)          ;HIGH ORDER 8 BITS ZEROED
1387 0801 C2EF          LD      LO(P2)
1388 0803 31            XPAL    P1              ;RESTORE ORIGINAL P1
1389 0804 40            LDE
1390 0805 35            XPAH    P1
1391 0806 90B0          JMP     X15
1392
1393
1394            ;***************************************
1395            ;*     MOVE -- STORE INTO MEMORY       *
1396            ;***************************************
1397
1398            ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
1399            ;     '@' FACTOR '=' REL-EXP
1400
1401 0808 C410 MOVE:   LDI     H(AESTK)
1402 080A 37            XPAH    P3
 403 080B C2FD          LD      LSTK(P2)
1404 080D 33            XPAL    P3              ;P3 -> ARITH STACK
```

```
1405 080E C7FE            LD      @-2(P3)          ;GET BYTE TO BE MOVED
1406 0810 01              XAE
1407 0811 C7FF            LD      @-1(P3)          ;NOW GET ADDRESS INTO P3
1408 0813 CAEA            ST      TEMP(P2)
1409 0815 C7FF            LD      @-1(P3)
1410 0817 33              XPAL    P3
1411 0818 CAFD            ST      LSTK(P2)         ;STACK PTR UPDATED NOW
1412 081A C2EA            LD      TEMP(P2)
1413 081C 37              XPAH    P3
1414 081D 40              LDE
1415 081E CB00            ST      0(P3)            ;MOVE THE BYTE INTO MEMORY
1416 0820 90C6    X17:    JMP     X16
1417 0822 90A6    E11:    JMP     E10
1418
1419                      ;*************************************
1420                      ;*              TEXT EDITOR          *
1421                      ;*************************************
1422
1423                      ;INPUTS TO THIS ROUTINE: POINTER TO LINE BUFFER IN P1LOW &
1424                      ; P1HIGH.  P1 POINTS TO THE INSERTION POINT IN THE TEXT.
1425                      ; THE A.E. STACK HAS THE LINE NUMBER ON IT (STACK POINTER
1426                      ; IS ALREADY POPPED).
1427
1428                      ;EACH LINE IN THE NIBL TEXT IS STORED IN THE FOLLOWING
1429                      ; FORMAT: TWO BYTES CONTAINING THE LINE NUMBER (IN BINARY,
1430                      ; HIGH ORDER BYTE FIRST), THEN ONE BYTE CONTAINING THE
1431                      ; LENGTH OF THE LINE, AND FINALLY THE LINE ITSELF FOLLOWED
1432                      ; BY A CARRIAGE RETURN.  THE LAST LINE IN THE TEXT IS
1433                      ; FOLLOWED BY TWO CONSECUTIVE BYTES OF X'FF.
1434
1435                              .LOCAL
1436 0824 C410    INSRT:  LDI     H(AESTK)         ;POINT P3 AT AE STACK,
1437 0826 37              XPAH    P3               ; WHICH HAS THE LINE #
1438 0827 C2FD            LD      LSTK(P2)         ; ON IT
1439 0829 33              XPAL    P3
1440 082A C301            LD      1(P3)            ;SAVE NEW LINE'S NUMBER
1441 082C CAF7            ST      HILINE(P2)
1442 082E C300            LD      0(P3)
1443 0830 CAF8            ST      LOLINE(P2)
1444 0832 C2F1            LD      P1LOW(P2)        ;PUT POINTER TO LBUF INTO P3
1445 0834 33              XPAL    P3
1446 0835 C2F0            LD      P1HIGH(P2)
1447 0837 37              XPAH    P3
1448 0838 C404            LDI     4                ;INITIALLY LENGTH OF NEW LINE
1449 083A CAE7            ST      CHRNUM(P2)       ; = 4.  ADD 1 TO LENGTH FOR
1450 083C C701    $1:     LD      @1(P3)           ; EACH CHAR IN LINE UP TO,
1451 083E E40D            XRI     0D               ; BUT NOT INCLUDING,
1452 0840 9804            JZ      $2               ; CARRIAGE RETURN
1453 0842 AAE7            ILD     CHRNUM(P2)
1454 0844 90F6            JMP     $1
1455 0846 C2E7    $2:     LD      CHRNUM(P2)       ;IF LENGTH STILL 4,
1456 0848 E404            XRI     4                ; WE'LL DELETE A LINE,
1457 084A 9C02            JNZ     $3               ; SO SET LENGTH = 0
1458 084C CAE7            ST      CHRNUM(P2)
```

```
1459 084E C2E7   $3:     LD      CHRNUM(P2)      ;PUT NEW LINE LENGTH IN EX
1460 0850 01             XAE
1461 0851 C2F2           LD      LABLHI(P2)      ;IS NEW LINE REPLACING OLD?
1462 0853 9406           JP      $4              ;YES - DO REPLACE
1463 0855 D47F           ANI     07F             ;NO - WE'LL INSERT LINE HERE,
1464 0857 CAF2           ST      LABLHI(P2)      ; WHERE FNDLBL GOT US
1465 0859 9018           JMP     $MOVE           ;BUT FIRST MAKE ROOM
1466 085B C503   $4:     LD      @3(P1)          ;SKIP LINE # AND LENGTH
1467 085D 40             LDE                     ;EX, NOW HOLDING NEW LINE
1468 085E 02             CCL                     ; LENGTH, WILL SOON HOLD
1469 085F F4FC           ADI     -4              ; DISPLACEMENT OF LINES
1470 0861 01             XAE                     ; TO BE MOVED
1471 0862 C501   $5:     LD      @1(P1)          ;SUBTRACT 1 FROM DISPLACEMENT
1472 0864 E40D           XRI     0D              ; FOR EACH CHAR IN LINE BEING
1473 0866 980B           JZ      $MOVE           ; REPLACED
1474 0868 40             LDE
1475 0869 02             CCL
1476 086A F4FF           ADI     -1
1477 086C 01             XAE
1478 086D 90F3           JMP     $5
1479 086F 90AF   X19:    JMP     X17
1480 0871 90AF   E12:    JMP     E11
1481 0873 40     $MOVE:  LDE                     ;IF DISPLACEMENT AND LENGTH
1482 0874 DAE7           OR      CHRNUM(P2)      ; OF NEW LINE ARE 0, RETURN
1483 0876 98F7           JZ      X19
1484 0878 C47A           LDI     L(DOSTAK)       ;CLEAR SOME STACKS
 485 087A CAFF           ST      DOPTR(P2)
1486 087C C46A           LDI     L(SBRSTK)
1487 087E CAFC           ST      SBRPTR(P2)
1488 0880 C48A           LDI     L(FORSTK)
1489 0882 CAFE           ST      FORPTR(P2)
1490 0884 40             LDE
1491 0885 9860           JZ      $ADD            ;DON'T NEED TO MOVE LINES
1492 0887 9410           JP      $UP             ;SKIP IF DISP. POSITIVE
1493 0889 C100   $DOWN:  LD      0(P1)           ;NEGATIVE DISPLACEMENT:
1494 088B C980           ST      EREG(P1)        ;DO;
1495 088D C501           LD      @1(P1)          ;    M(P1+DISP) = M(P1);
1496 088F 94F8           JP      $DOWN           ;    P1 = P1+1;
1497 0891 C100           LD      0(P1)           ;UNTIL M(P1)<0 & M(P1-1)<0;
1498 0893 94F4           JP      $DOWN
1499 0895 C980           ST      EREG(P1)        ;M(P1+DISP) = M(P1);
1500 0897 904E           JMP     $ADD
1501 0899 C1FE   $UP:    LD      -2(P1)          ;POSITIVE DISPLACEMENT:
1502 089B CAEA           ST      TEMP(P2)        ;FLAG BEGINNING OF MOVE WITH
1503 089D C4FF           LDI     -1              ; A -1 FOLLOWED BY 80, WHICH
1504 089F C9FE           ST      -2(P1)          ; CAN NEVER APPEAR IN A
1505 08A1 C450           LDI     80              ; NIBL TEXT
1506 08A3 C9FF           ST      -1(P1)
1507 08A5 C501   $UP1:   LD      @1(P1)          ;ADVANCE P1 TO END OF TEXT
1508 08A7 94FC           JP      $UP1
1509 08A9 C100           LD      0(P1)
1510 08AB 94F8           JP      $UP1
 511 08AD 35             XPAH    P1              ;SAVE P1 IN LO, HI
1512 08AE CAEE           ST      HI(P2)
```

```
1513 08B0 35              XPAH    P1
1514 08B1 31              XPAL    P1
1515 08B2 CAEF            ST      LO(P2)
1516 08B4 31              XPAL    P1
1517 08B5 C2EF            LD      LO(P2)          ;ADD DISPLACEMENT TO
1518 08B7 02              CCL                     ; VALUE OF P1, TO CHECK
1519 08B8 70              ADE                     ; WHETHER WE'RE OUT OF
1520 08B9 C400            LDI     0               ; RAM FOR USER'S PROGRAM
1521 08BB F2EE            ADD     HI(P2)
1522 08BD E2EE            XOR     HI(P2)
1523 08BF D4F0            ANI     0F0
1524 08C1 9803            JZ      $UP2
1525 08C3 C400            LDI     0               ;IF OUT OF RAM, CHANGE
1526 08C5 01              XAE                     ; DISPLACEMENT TO ZERO
1527 08C6 C4FF    $UP2:   LDI     -1
1528 08C8 C980    $UP3:   ST      EREG(P1)        ;MOVE TEXT UP UNTIL WE REACH
1529 08CA C5FF            LD      @-1(P1)         ; THE FLAGS SET ABOVE
1530 08CC 94FA            JP      $UP3
1531 08CE C101            LD      1(P1)
1532 08D0 E450            XRI     80
1533 08D2 9804            JZ      $UP4
1534 08D4 C100            LD      0(P1)
1535 08D6 90F0            JMP     $UP3
1536 08D8 C2EA    $UP4:   LD      TEMP(P2)        ;RESTORE THE FLAGGED LOCATION
1537 08DA C900            ST      0(P1)           ; TO THEIR ORIGINAL VALUES
1538 08DC C40D            LDI     0D
1539 08DE C901            ST      1(P1)
1540 08E0 40              LDE                     ;IF DISPLACEMENT = 0, WE'RE
1541 08E1 9C04            JNZ     $ADD            ; OUT OF RAM, SO REPORT ERROR
1542 08E3 C402            LDI     2
1543 08E5 908A    E12A:   JMP     E12
1544 08E7 C2E7    $ADD:   LD      CHRNUM(P2)      ;INSERT NEW LINE
1545 08E9 9884    X19A:   JZ      X19             ; UNLESS LENGTH IS ZERO
1546 08EB C2F1            LD      P1LOW(P2)       ;POINT P1 AT LINE BUFFER
1547 08ED 31              XPAL    P1
1548 08EE C2F0            LD      P1HIGH(P2)
1549 08F0 35              XPAH    P1
1550 08F1 C2F3            LD      LABLLO(P2)      ;POINT P3 AT INSERTION PLACE
1551 08F3 33              XPAL    P3
1552 08F4 C2F2            LD      LABLHI(P2)
1553 08F6 37              XPAH    P3
1554 08F7 C2F7            LD      HILINE(P2)      ;PUT LINE NUMBER INTO TEXT
1555 08F9 CF01            ST      @1(P3)
1556 08FB C2F8            LD      LOLINE(P2)
1557 08FD CF01            ST      @1(P3)
1558 08FF C2E7            LD      CHRNUM(P2)      ;STORE LINE LENGTH IN TEXT
1559 0901 CF01            ST      @1(P3)
1560 0903 C501    $ADD1:  LD      @1(P1)          ;PUT REST OF CHARS
1561 0905 CF01            ST      @1(P3)          ; (INCLUDING CR) INTO TEXT
1562 0907 E40D            XRI     0D
1563 0909 9CF8            JNZ     $ADD1
1564 090B 90DC            JMP     X19A            ;RETURN
1565 090D C400    X20:    JS      P3,EXECIL
1566 0914 90CF    E13:    JMP     E12A
```

```
1567
1568
1569                ;*************************************
1570                ;*        POP ARITHMETIC STACK       *
1571                ;*************************************
1572
1573 0916 BAFD   POPAE:  DLD     LSTK(P2)        ;THIS ROUTINE POP THE A.E.
1574 0918 BAFD           DLD     LSTK(P2)        ; STACK, AND PUTS THE RESULT
1575 091A 33             XPAL    P3              ; INTO LO(P2) AND HI(P2)
1576 091B C410           LDI     H(AESTK)
1577 091D 37             XPAH    P3
1578 091E C300           LD      (P3)
1579 0920 CAEF           ST      LO(P2)
1580 0922 C301           LD      1(P3)
1581 0924 CAEE           ST      HI(P2)
1582 0926 90E5           JMP     X20
1583
1584
1585                ;*************************************
1586                ;*                UNTIL              *
1587                ;*************************************
1588
1589                      .LOCAL
1590 0928 C2FF   UNTIL:  LD      DOPTR(P2)       ;CHECK FOR DO-STACK UNDERFLOW
1591 092A 01             XAE
1592 092B 40             LDE
1593 092C E47A           XRI     L(DOSTAK)
1594 092E 9C04           JNZ     $1
1595 0930 C40F           LDI     15
1596 0932 90E0           JMP     E13
1597 0934 C2EF   $1:     LD      LO(P2)          ;CHECK FOR EXPRESSION = 0
1598 0936 DAEE           OR      HI(P2)
1599 0938 9806           JZ      $REDO           ;IF ZERO, REPEAT DO-LOOP
1600 093A BAFF           DLD     DOPTR(P2)       ;ELSE POP SAVE STACK
1601 093C BAFF           DLD     DOPTR(P2)
1602 093E 90CD           JMP     X20             ;CONTINUE TO NEXT STMT
1603 0940 40     $REDO:  LDE                     ;POINT P3 AT DO-STACK
1604 0941 33             XPAL    P3
1605 0942 C410           LDI     H(DOSTAK)
1606 0944 37             XPAH    P3
1607 0945 C3FF           LD      -1(P3)          ;LOAD P1 FROM DO STACK
1608 0947 35             XPAH    P1
1609 0948 C3FE           LD      -2(P3)
1610 094A 31             XPAL    P1              ;CURSOR NOW POINTS TO FIRST
1611 094B 90C0           JMP     X20             ; STATEMENT OF DO-LOOP
1612
1613
1614                ;*************************************
1615                ;*     STORE INTO STATUS REGISTER    *
1616                ;*************************************
1617
1618                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
1619                ;      'STAT' '=' REL-EXP
1620
```

```
1621 094D C2EF   MOVESR: LD      LO(P2)          ;LOW BYTE GOES TO STATUS
1622 094F D4F7           ANI     0F7             ; BUT WITH IEN BIT CLEARED
1623 0951 07             CAS
1624 0952 90B9   X21:    JMP     X20
1625 0954 90BE   E14:    JMP     E13
1626
1627
1628 }            ;****************************************
1629              ;*          STAT FUNCTION               *
1630              ;****************************************
1631
1632 0956 C410   STATUS: LDI     H(AESTK)
1633 0958 37             XPAH    P3              ;POINT P3 AT AE STACK
1634 0959 AAFD           ILD     LSTK(P2)
1635 095B AAFD           ILD     LSTK(P2)
1636 095D 33             XPAL    P3
1637 095E 06             CSA
1638 095F CBFE           ST      -2(P3)          ;STATUS REG IS LOW BYTE
1639 0961 C400           LDI     0
1640 0963 CBFF           ST      -1(P3)          ;ZERO IS HIGH BYTE
1641 0965 90EB           JMP     X21
1642
1643
1644              ;****************************************
1645              ;*    MACHINE LANGUAGE SUBROUTINE       *
1646              ;****************************************
1647
1648              ; THIS ROUTINE IMPLEMENTS THE 'LINK' STATEMENT
1649
1650 0967 C2EE   CALLML: LD      HI(P2)          ;GET HIGH BYTE OF ADDRESS
1651 0969 37             XPAH    P3
1652 096A C2EF           LD      LO(P2)          ;GET LOW BYTE
1653 096C 33             XPAL    P3              ;P3 -> USER'S ROUTINE
1654 096D C7FF           LD      @-1(P3)         ;CORRECT P3
1655 096F 3F             XPPC    P3              ;CALL ROUTINE (PRAY IT WORKS)
1656 0970                LDPI    P2,VARS         ;RESTORE RAM POINTER
1657 0976 90DA           JMP     X21             ;RETURN
1658
1659
1660              ;****************************************
1661              ;*         SAVE DO LOOP ADDRESS         *
1662              ;****************************************
1663
1664              ; THIS ROUTINE IMPLEMENTS THE 'DO' STATEMENT.
1665
1666                       .LOCAL
1667 0978 C2FF   SAVEDO: LD      DOPTR(P2)       ;CHECK FOR STACK OVERFLOW
1668 097A E48A           XRI     L(FORSTK)
1669 097C 9C04           JNZ     $1
1670 097E C40A           LDI     10
1671 0980 90D2   E15:    JMP     E14
1672 0982 AAFF   $1:     ILD     DOPTR(P2)
1673 0984 AAFF           ILD     DOPTR(P2)
1674 0986 33             XPAL    P3
```

```
1675 0987 C410            LDI     H(DOSTAK)
1676 0989 37              XPAH    P3              ;P3 -> TOP OF DO STACK
1677 098A 35              XPAH    P1              ;SAVE CURSOR ON THE STACK
1678 098B CBFF            ST      -1(P3)
1679 098D 35              XPAH    P1
1680 098E 31              XPAL    P1
1681 098F CBFE            ST      -2(P3)
1682 0991 31              XPAL    P1
1683 0992 90BE    X22:    JMP     X21
1684
1685
1686              ;****************************************
1687              ;*         TOP OF RAM FUNCTION          *
1688              ;****************************************
1689
1690                      .LOCAL
1691 0994 C2E9    TOP:    LD      TEMP2(P2)       ;SET P3 TO POINT TO
1692 0996 37              XPAH    P3              ; START OF NIBL TEXT
1693 0997 C2E8            LD      TEMP3(P2)
1694 0999 33              XPAL    P3
1695 099A C300    $0:     LD      (P3)            ;HAVE WE HIT END OF TEXT?
1696 099C 9402            JP      $1              ;NO - SKIP TO NEXT LINE
1697 099E 9007            JMP     $2              ;YES - PUT CURSOR ON STACK
1698 09A0 C302    $1:     LD      2(P3)           ;GET LENGTH OF LINE
1699 09A2 01              XAE
 700 09A3 C780            LD      @EREG(P3)       ;SKIP TO NEXT LINE
1701 09A5 90F3            JMP     $0              ;GO CHECK FOR EOF
1702 09A7 C702    $2:     LD      @2(P3)          ;P3 := P3 + 2
1703 09A9 AAFD            ILD     LSTK(P2)        ;SET P3 TO STACK, SAVING
1704 09AB AAFD            ILD     LSTK(P2)        ; OLD P3 (WHICH CONTAINS TOP)
1705 09AD 33              XPAL    P3              ; ON IT SOMEHOW
1706 09AE 01              XAE
1707 09AF C410            LDI     H(AESTK)
1708 09B1 37              XPAH    P3
1709 09B2 CBFF            ST      -1(P3)
1710 09B4 40              LDE
1711 09B5 CBFE            ST      -2(P3)
1712 09B7 90D9            JMP     X22
1713
1714
1715              ;****************************************
1716              ;*       SKIP TO NEXT NIBL LINE         *
1717              ;****************************************
1718
1719 09B9 C501    IGNORE: LD      @1(P1)          ;SCAN TIL WE'RE PAST
1720 09BB E40D            XRI     0D              ; CARRIAGE RETURN
1721 09BD 9CFA            JNZ     IGNORE
1722 09BF 3F              XPPC    P3
1723
1724
1725              ;****************************************
 726              ;*           MODULO FUNCTION            *
 727              ;****************************************
1728
```

```
1729 09C0 C2FD   MODULO: LD      LSTK(P2)        ;THIS ROUTINE MUST BE
1730 09C2 33             XPAL    P3              ;  IMMEDIATELY AFTER A
1731 09C3 C410           LDI     H(AESTK)        ;  DIVIDE TO WORK CORRECTLY
1732 09C5 37             XPAH    P3
1733 09C6 C303           LD      3(P3)           ;GET LOW BYTE OF REMAINDER
1734 09C8 CBFE           ST      -2(P3)          ;PUT ON STACK
1735 09CA C302           LD      2(P3)           ;GET HIGH BYTE OF REMAINDER
1736 09CC CBFF           ST      -1(P3)          ;PUT ON STACK
1737 09CE 90C2   X23:    JMP     X22
1738 09D0 90AE   E16:    JMP     E15
1739
1740
1741             ;****************************************
1742             ;*           RANDOM FUNCTION            *
1743             ;****************************************
1744
1745                       .LOCAL
1746 09D2 C408   RANDOM: LDI     8               ;LOOP COUNTER FOR MULTIPLY
1747 09D4 CAEB           ST      NUM(P2)
1748 09D6 C2E5           LD      RNDX(P2)
1749 09D8 01             XAE
1750 09D9 C2E4           LD      RNDY(P2)
1751 09DB CAE9           ST      TEMP2(P2)
1752 09DD C2E5   $LOOP:  LD      RNDX(P2)        ;MULTIPLY THE SEEDS BY 9
1753 09DF 02             CCL
1754 09E0 70             ADE
1755 09E1 01             XAE
1756 09E2 C2E4           LD      RNDY(P2)
1757 09E4 02             CCL
1758 09E5 F2E9           ADD     TEMP2(P2)
1759 09E7 CAE4           ST      RNDY(P2)
1760 09E9 BAEB           DLD     NUM(P2)
1761 09EB 9CF0           JNZ     $LOOP
1762 09ED 40             LDE                     ;ADD 7 TO SEEDS
1763 09EE 02             CCL
1764 09EF F407           ADI     7
1765 09F1 01             XAE
1766 09F2 C2E4           LD      RNDY(P2)
1767 09F4 02             CCL
1768 09F5 F407           ADI     7
1769 09F7 1E             RR
1770 09F8 CAE4           ST      RNDY(P2)
1771 09FA AAE6           ILD     RNDF(P2)        ;HAVE WE GONE THROUGH
1772 09FC 9803           JZ      $1              ;  256 GENERATIONS?
1773 09FE 40             LDE                     ;IF SO, SKIP GENERATING
1774 09FF CAE5           ST      RNDX(P2)        ;  THE NEW RNDX
1775 0A01 C2FD   $1:     LD      LSTK(P2)        ;START MESSING WITH THE STACK
1776 0A03 33             XPAL    P3
1777 0A04 C410           LDI     H(AESTK)
1778 0A06 37             XPAH    P3
1779 0A07 C401           LDI     1               ;FIRST PUT 1 ON STACK
1780 0A09 CB00           ST      (P3)
1781 0A0B C400           LDI     0
1782 0A0D CB01           ST      1(P3)
```

```
1783 0A0F C3FE            LD      -2(P3)          ;PUT EXPR2 ON STACK
1784 0A11 CB02            ST      2(P3)
1785 0A13 C3FF            LD      -1(P3)
1786 0A15 CB03            ST      3(P3)
1787 0A17 C3FC            LD      -4(P3)          ;PUT EXPR1 ON STACK
1788 0A19 CB04            ST      4(P3)
1789 0A1B C3FD            LD      -3(P3)
1790 0A1D CB05            ST      5(P3)
1791 0A1F C2E4            LD      RNDY(P2)        ;PUT RANDOM # ON STACK
1792 0A21 CBFE            ST      -2(P3)
1793 0A23 C2E5            LD      RNDX(P2)
1794 0A25 E4FF            XRI     0FF
1795 0A27 D47F            ANI     07F
1796 0A29 CBFF            ST      -1(P3)
1797 0A2B C706            LD      @6(P3)          ;ADD 6 TO STACK POINTER
1798 0A2D 33              XPAL    P3
1799 0A2E CAFD            ST      LSTK(P2)
1800 0A30 909C    X24:    JMP     X23
1801 0A32 909C    E16A:   JMP     E16
1802
1803
1804              ;***************************************
1805              ;*      PUSH 1 ON ARITHMETIC STACK     *
1806              ;***************************************
1807
1808 0A34 AAFD    LIT1:   ILD     LSTK(P2)
1809 0A36 AAFD            ILD     LSTK(P2)
1810 0A38 33              XPAL    P3
1811 0A39 C410            LDI     H(AESTK)
1812 0A3B 37              XPAH    P3
1813 0A3C C400            LDI     0
1814 0A3E CBFF            ST      -1(P3)
1815 0A40 C401            LDI     1
1816 0A42 CBFE            ST      -2(P3)
1817 0A44 90EA            JMP     X24
1818
1819
1820              ;***************************************
1821              ;*      FOR-LOOP INITIALIZATION        *
1822              ;***************************************
1823
1824                      .LOCAL
1825 0A46 C2FE    SAVFOR: LD      FORPTR(P2)      ;CHECK FOR FOR STACK
1826 0A48 E4A6            XRI     L(PCSTAK)       ; OVERFLOW
1827 0A4A 9C04            JNZ     $1
1828 0A4C C40A            LDI     10
1829 0A4E 90E2    E17:    JMP     E16A
1830 0A50 E4A6    $1:     XRI     L(PCSTAK)
1831 0A52 31              XPAL    P1              ;POINT P1 AT FOR STACK
1832 0A53 CAF1            ST      P1LOW(P2)       ; SAVING OLD P1
1833 0A55 C410            LDI     H(FORSTK)
1834 0A57 35              XPAH    P1
1835 0A58 CAF0            ST      P1HIGH(P2)
1836 0A5A C2FD            LD      LSTK(P2)        ;POINT P3 AT AE STACK
```

```
1837 0A5C 33              XPAL    P3
1838 0A5D C410            LDI     H(AESTK)
1839 0A5F 37              XPAH    P3
1840 0A60 C3F9            LD      -7(P3)          ;GET VARIABLE INDEX
1841 0A62 CD01            ST      @1(P1)          ;SAVE ON FOR-STACK
1842 0A64 C3FC            LD      -4(P3)          ;GET L(LIMIT)
1843 0A66 CD01            ST      @1(P1)          ;SAVE
1844 0A68 C3FD            LD      -3(P3)          ;GET H(LIMIT)
1845 0A6A CD01            ST      @1(P1)          ;SAVE
1846 0A6C C3FE            LD      -2(P3)          ;GET L(STEP)
1847 0A6E CD01            ST      @1(P1)          ;SAVE
1848 0A70 C3FF            LD      -1(P3)          ;GET H(STEP)
1849 0A72 CD01            ST      @1(P1)          ;SAVE
1850 0A74 C2F1            LD      P1LOW(P2)       ;GET L(P1)
1851 0A76 CD01            ST      @1(P1)          ;SAVE
1852 0A78 C2F0            LD      P1HIGH(P2)      ;GET H(P1)
1853 0A7A CD01            ST      @1(P1)          ;SAVE
1854 0A7C 35              XPAH    P1              ;RESTORE OLD P1
1855 0A7D C2F1            LD      P1LOW(P2)
1856 0A7F 31              XPAL    P1
1857 0A80 CAFE            ST      FORPTR(P2)      ;UPDATE FOR STACK PTR
1858 0A82 C7FC            LD      @-4(P3)
1859 0A84 33              XPAL    P3
1860 0A85 CAFD            ST      LSTK(P2)        ;UPDATE AE STACK PTR
1861 0A87 90A7    X25:    JMP     X24
1862
1863
1864              ;***************************************
1865              ;*      FIRST PART OF 'NEXT VAR'       *
1866              ;***************************************
1867
1868                      .LOCAL
1869 0A89 C2FE    NEXTV:  LD      FORPTR(P2)      ;POINT P1 AT FOR STACK,
1870 0A8B E48A            XRI     L(FORSTK)       ;  CHECKING FOR UNDERFLOW
1871 0A8D 9C04            JNZ     $1
1872 0A8F C40B            LDI     11              ;REPORT ERROR
1873 0A91 90BB            JMP     E17
1874 0A93 E48A    $1:     XRI     L(FORSTK)
1875 0A95 31              XPAL    P1
1876 0A96 CAF1            ST      P1LOW(P2)       ;SAVE OLD P1
1877 0A98 C410            LDI     H(FORSTK)
1878 0A9A 35              XPAH    P1
1879 0A9B CAF0            ST      P1HIGH(P2)
1880 0A9D C2FD            LD      LSTK(P2)        ;POINT P3 AT AE STACK
1881 0A9F 33              XPAL    P3
1882 0AA0 C410            LDI     H(AESTK)
1883 0AA2 37              XPAH    P3
1884 0AA3 C7FF            LD      @-1(P3)         ;GET VARIABLE INDEX
1885 0AA5 E1F9            XOR     -7(P1)          ;COMPARE WITH INDEX
1886 0AA7 9804            JZ      $10             ;  ON FOR STACK: ERROR
1887 0AA9 C40C            LDI     12              ;  IF NOT EQUAL
1888 0AAB 90A1    E18:    JMP     E17
1889 0AAD E1F9    $10:    XOR     -7(P1)          ;RESTORE INDEX
1890 0AAF 01              XAE                     ;SAVE IN EREG
```

```
1891 0AB0 C280          LD      EREG(P2)        ;GET L(VARIABLE)
1892 0AB2 02            CCL
1893 0AB3 F1FC          ADD     -4(P1)          ;ADD L(STEP)
1894 0AB5 CA80          ST      EREG(P2)        ;STORE IN VARIABLE
1895 0AB7 CB00          ST      (P3)            ; AND ON STACK
1896 0AB9 C601          LD      @1(P2)          ;INCREMENT RAM PTR
1897 0ABB C280          LD      EREG(P2)        ;GET H(VARIABLE)
1898 0ABD F1FD          ADD     -3(P1)          ;ADD H(STEP)
1899 0ABF CA80          ST      EREG(P2)        ;STORE IN VARIABLE
1900 0AC1 CB01          ST      1(P3)           ; AND ON STACK
1901 0AC3 C6FF          LD      @-1(P2)         ;RESTORE RAM POINTER
1902 0AC5 C1FA          LD      -6(P1)          ;GET L(LIMIT)
1903 0AC7 CB02          ST      2(P3)           ;PUT ON STACK
1904 0AC9 C1FB          LD      -5(P1)          ;GET H(LIMIT)
1905 0ACB CB03          ST      3(P3)           ;PUT ON STACK
1906 0ACD C1FD          LD      -3(P1)          ;GET H(STEP)
1907 0ACF 9410          JP      $2              ;IF NEGATIVE, INVERT
1908 0AD1 C404          LDI     4               ; ITEMS ON A.E. STACK
1909 0AD3 CAEB          ST      NUM(P2)         ;NUM = LOOP COUNTER
1910 0AD5 C701  $LOOP:  LD      @1(P3)          ;GET BYTE FROM STACK
1911 0AD7 E4FF          XRI     0FF             ;INVERT IT
1912 0AD9 CBFF          ST      -1(P3)          ;PUT BACK ON STACK
1913 0ADB BAEB          DLD     NUM(P2)         ;DO UNTIL NUM = 0
1914 0ADD 9CF6          JNZ     $LOOP
 915 0ADF 9002          JMP     $3
1916 0AE1 C704  $2:     LD      @4(P3)          ;UPDATE AE STACK POINTER
1917 0AE3 33    $3;     XPAL    P3
1918 0AE4 CAFD          ST      LSTK(P2)
1919 0AE6 C2F1          LD      P1LOW(P2)       ;RESTORE OLD P1
1920 0AE8 31            XPAL    P1
1921 0AE9 C2F0          LD      P1HIGH(P2)
1922 0AEB 35            XPAH    P1
1923 0AEC 9099  X26:    JMP     X25
1924
1925
1926                    ;****************************************
1927                    ;*      SECOND PART OF 'NEXT VAR'       *
1928                    ;****************************************
1929
1930  0AEE C2EF   NEXTV1: LD      LO(P2)          ;IS FOR-LOOP OVER WITH?
1931 0AF0 9808          JZ      $REDO           ;NO - REPEAT LOOP
1932 0AF2 C2FE          LD      FORPTR(P2)      ;YES - POP FOR-STACK
1933 0AF4 02            CCL
1934 0AF5 F4F9          ADI     -7
1935 0AF7 CAFE          ST      FORPTR(P2)
1936 0AF9 3F            XPPC    P3              ;RETURN TO I.L. INTERPRETER
1937 0AFA C2FE  $REDO:  LD      FORPTR(P2)      ;POINT P3 AT FOR STACK
1938 0AFC 33            XPAL    P3
1939 0AFD C410          LDI     H(FORSTK)
1940 0AFF 37            XPAH    P3
 941 0B00 C3FF          LD      -1(P3)          ;GET OLD P1 OFF STACK
 942 0B02 35            XPAH    P1
1943 0B03 C3FE          LD      -2(P3)
1944 0B05 31            XPAL    P1
```

```
0B06 90E4            JMP     X26
0B08 90A1   E19:     JMP     E18

                ;*************************************
                ;*      PRINT MEMORY AS STRING       *
                ;*************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;      'PRINT' '$' FACTOR

                     .LOCAL
0B0A C2EE   PSTRNG:  LD      HI(P2)          ;POINT P1 AT STRING TO PRINT
0B0C 35              XPAH    P1
0B0D C2EF            LD      LO(P2)
0B0F 31              XPAL    P1
0B10                 LDPI    P3,PUTC-1       ;POINT P3 AT PUTC ROUTINE
0B16 C501   $1:      LD      @1(P1)          ;GET A CHARACTER
0B18 E40D            XRI     0D              ;IS IT A CARRIAGE RETURN?
0B1A 98D0            JZ      X26             ;YES - WE'RE DONE
0B1C E40D            XRI     0D              ;NO - PRINT THE CHARACTER
0B1E 3F              XPPC    P3
0B1F 06              CSA                     ;MAKE SURE NO ONE IS
0B20 D420            ANI     020             ; TYPING ON THE TTY
0B22 9CF2            JNZ     $1              ; BEFORE REPEATING LOOP
0B24 90C6            JMP     X26

                ;*************************************
                ;*         INPUT A STRING            *
                ;*************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;      'INPUT' '$' FACTOR

0B26 C2EE   ISTRNG:  LD      HI(P2)          ;GET ADDRESS TO STORE THE
0B28 37              XPAH    P3              ; STRING, PUT IT INTO P3
0B29 C2EF            LD      LO(P2)
0B2B 33              XPAL    P3
0B2C C501   $2:      LD      @1(P1)          ;GET A BYTE FROM LINE BUFFER
0B2E CF01            ST      @1(P3)          ;PUT IT IN SPECIFIED LOCATION
0B30 E40D            XRI     0D              ;DO UNTIL CHAR = CARR. RETURN
0B32 9CF8            JNZ     $2
0B34 90B6   X27:     JMP     X26

                ;*************************************
                ;*   STRING CONSTANT ASSIGNMENT      *
                ;*************************************

                ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
                ;      '$' FACTOR '=' STRING

                     .LOCAL
```

```
1999 0B36 C2EF   PUTSTR: LD      LO(P2)          ;GET ADDRESS TO STORE STRING,
2000 0B38 33             XPAL    P3              ; PUT IT INTO P3
2001 0B39 C2EE           LD      HI(P2)
2002 0B3B 37             XPAH    P3
2003 0B3C C501   $LOOP:  LD      @1(P1)          ;GET A BYTE FROM STRING
2004 0B3E E422           XRI     '"'             ;CHECK FOR END OF STRING
2005 0B40 980E           JZ      $END
2006 0B42 E42F           XRI     '"' !.OD        ;MAKE SURE THERE'S NO CR
2007 0B44 9C04           JNZ     $1
2008 0B46 C407           LDI     7
2009 0B48 90BE           JMP     E19             ;ERROR IF CARRIAGE RETURN
2010 0B4A E40D   $1:     XRI     OD              ;RESTORE CHARACTER
2011 0B4C CF01           ST      @1(P3)          ;PUT IN SPECIFIED LOCATION
2012 0B4E 90EC           JMP     $LOOP           ;GET NEXT CHARACTER
2013 0B50 C40D   $END:   LDI     OD              ;APPEND CARRIAGE RETURN
2014 0B52 CB00           ST      (P3)            ; TO STRING
2015 0B54 90DE           JMP     X27
2016
2017
2018             ;*************************************
2019             ;*            MOVE STRING            *
2020             ;*************************************
2021
 022             ; THIS ROUTINE IMPLEMENTS THE STATEMENT:
2023             ;      '$' FACTOR '=' '$' FACTOR
2024
2025                     .LOCAL
2026 0B56 C2FD   MOVSTR: LD      LSTK(P2)        ;POINT P3 AT A.E. STACK
2027 0B58 33             XPAL    P3
2028 0B59 C410           LDI     H(AESTK)
2029 0B5B 37             XPAH    P3
2030 0B5C C7FF           LD      @-1(P3)         ;GET ADDRESS OF SOURCE STRING
2031 0B5E 35             XPAH    P1              ; INTO P1
2032 0B5F C7FF           LD      @-1(P3)
2033 0B61 31             XPAL    P1
2034 0B62 C7FF           LD      @-1(P3)         ;GET ADDRESS OF DESTINATION
2035 0B64 01             XAE                     ; STRING INTO P3
2036 0B65 C7FF           LD      @-1(P3)
2037 0B67 33             XPAL    P3
2038 0B68 CAFD           ST      LSTK(P2)        ;UPDATE STACK POINTER
2039 0B6A 40             LDE
2040 0B6B 37             XPAH    P3
2041 0B6C C501   $LOOP:  LD      @1(P1)          ;GET A SOURCE CHARACTER
2042 0B6E CF01           ST      @1(P3)          ;SEND IT TO DESTINATION
2043 0B70 E40D           XRI     OD              ;REPEAT UNTIL CARRIAGE RET.
2044 0B72 98C0           JZ      X27
2045 0B74 06             CSA                     ; OR KEYBOARD INTERRUPT
2046 0B75 D420           ANI     020
2047 0B77 9CF3           JNZ     $LOOP
 048 0B79 90B9           JMP     X27
2049
2050
2051             ;*************************************
2052             ;*     PUT PAGE NUMBER ON STACK      *
```

```
2053                ;*************************************
2054
2055 0B7B AAFD   PUTPGE: ILD      LSTK(P2)
2056 0B7D AAFD           ILD      LSTK(P2)
2057 0B7F 33             XPAL     P3
2058 0B80 C410           LDI      H(AESTK)
2059 0B82 37             XPAH     P3
2060 0B83 C2F6           LD       PAGE(P2)
2061 0B85 CBFE           ST       -2(P3)
2062 0B87 C400           LDI      0
2063 0B89 CBFF           ST       -1(P3)
2064 0B8B 90A7           JMP      X27
2065
2066
2067                ;*************************************
2068                ;*         ASSIGN NEW PAGE           *
2069                ;*************************************
2070
2071                         .LOCAL
2072 0B8D C2EF   NUPAGE: LD       LD(P2)           ;GET PAGE # FROM STACK,
2073 0B8F D407           ANI      7                ; GET THE LOW 3 BITS
2074 0B91 9C02           JNZ      $0               ;PAGE 0 BECOMES PAGE 1
2075 0B93 C401           LDI      1
2076 0B95 CAF6   $0:     ST       PAGE(P2)
2077 0B97 3F             XPPC     P3               ;RETURN
2078
2079
2080                ;*************************************
2081                ;*          FIND START OF PAGE       *
2082                ;*************************************
2083
2084                ; THIS ROUTINE COMPUTES THE START OF THE CURRENT TEXT PAGE,
2085                ; STORING THE ADDRESS IN TEMP2(P2) [THE HIGH BYTE], AND
2086                ; TEMP3(P2) [THE LOW BYTE].
2087
2088 0B98 C2F6   FNDPGE: LD       PAGE(P2)
2089 0B9A E401           XRI      1                ;SPECIAL CASE IS PAGE 1, BUT
2090 0B9C 9C09           JNZ      $1               ; OTHERS ARE CONVENTIONAL
2091 0B9E C411           LDI      H(PGM)           ;PAGE 1 STARTS AT 'PGM'
2092 0BA0 CAE9           ST       TEMP2(P2)
2093 0BA2 C420           LDI      L(PGM)
2094 0BA4 CAE8           ST       TEMP3(P2)
2095 0BA6 3F             XPPC     P3               ;RETURN
2096 0BA7 E401   $1:     XRI      1                ;RESTORE PAGE #
2097 0BA9 01             XAE                       ;SAVE IT
2098 0BAA C404           LDI      4                ;LOOP COUNTER = 4
2099 0BAC CAEB           ST       NUM(P2)
2100 0BAE 40     $LOOP:  LDE                       ;MULTIPLY PAGE# BY 16
2101 0BAF 02             CCL
2102 0BB0 70             ADE
2103 0BB1 01             XAE
2104 0BB2 BAEB           DLD      NUM(P2)
2105 0BB4 9CF8           JNZ      $LOOP
2106 0BB6 40             LDE
```

```
2107 0BB7 CAE9           ST      TEMP2(P2)       ;TEMP2 HAS HIGH BYTE
2108 0BB9 C402           LDI     2               ; OF ADDRESS NOW
2109 0BBB CAE8           ST      TEMP3(P2)       ;LOW BYTE IS ALWAYS 2
2110 0BBD 3F             XPPC    P3
2111
2112
2113             ;*************************************
2114             ;*      MOVE CURSOR TO NEW PAGE      *
2115             ;*************************************
2116
2117 0BBE C2E9   CHPAGE:  LD      TEMP2(P2)       ;PUT START OF PAGE
2118 0BC0 35             XPAH    P1              ; INTO P1.   THIS ROUTINE
2119 0BC1 C2E8           LD      TEMP3(P2)       ; MUST BE CALLED RIGHT
2120 0BC3 31             XPAL    P1              ; AFTER 'FNDPGE'
2121 0BC4 3F             XPPC    P3              ;RETURN
2122
2123
2124             ;*************************************
2125             ;*      DETERMINE CURRENT PAGE       *
2126             ;*************************************
2127
2128 0BC5 35     DETPGE: XPAH    P1              ;CURRENT PAGE IS HIGH
2129 0BC6 01             XAE                     ; PART OF CURSOR DIVIDED
2130 0BC7 40             LDE                     ; BY 16
2131 0BC8 35             XPAH    P1
2132 0BC9 40             LDE
2133 0BCA 1C             SR
2134 0BCB 1C             SR
2135 0BCC 1C             SR
2136 0BCD 1C             SR
2137 0BCE CAF6           ST      PAGE(P2)
2138 0BD0 3F             XPPC    P3              ;RETURN
2139
2140
2141             ;*************************************
2142             ;*          CLEAR CURRENT PAGE       *
2143             ;*************************************
2144
2145 0BD1 C2E9   NEWPGM: LD      TEMP2(P2)       ;POINT P1 AT CURRENT PAGE
2146 0BD3 35             XPAH    P1
2147 0BD4 C2E8           LD      TEMP3(P2)
2148 0BD6 31             XPAL    P1
2149 0BD7 C40D           LDI     0D              ;PUT DUMMY END-OF-LINE
2150 0BD9 C9FF           ST      -1(P1)          ; JUST BEFORE TEXT
2151 0BDB C4FF           LDI     -1              ;PUT -1 AT START OF TEXT
2152 0BDD C900           ST      (P1)
2153 0BDF C901           ST      1(P1)
2154 0BE1 3F             XPPC    P3              ;RETURN
2155
2156
2157             ;*************************************
2158             ;*      FIND LINE NUMBER IN TEXT     *
2159             ;*************************************
2160
```

```
2161                ;INPUTS:  THE START OF THE CURRENT PAGE IN TEMP2 AND TEMP3,
2162                ;         THE LINE NUMBER TO LOOK FOR IN LO AND HI.
2163                ;OUTPUTS: THE ADDRESS OF THE FIRST LINE IN THE NIBL TEXT
2164                ;         WHOSE LINE NUMBER IS GREATER THAN OR EQUAL TO THE
2165                ;         NUMBER IN HI AND LO, RETURNED IN P1 AND ALSO IN
2166                ;         IN THE RAM VARIABLES LABLLO AND LABLHI.  THE SIGN
2167                ;         BIT OF LABLHI IS SET IF EXACT LINE IS NOT FOUND.
2168
2169                         .LOCAL
2170 0BE2 C2E9    FNDLBL:  LD      TEMP2(P2)        ;POINT P1 AT START OF TEXT
2171 0BE4 35               XPAH    P1
2172 0BE5 C2E8             LD      TEMP3(P2)
2173 0BE7 31               XPAL    P1
2174 0BE8 C100    $1:      LD      (P1)             ;HAVE WE HIT END OF TEXT?
2175 0BEA E4FF             XRI     OFF
2176 0BEC 9412             JP      $2               ;YES - STOP LOOKING
2177 0BEE 03               SCL                      ;NO - COMPARE LINE NUMBERS
2178 0BEF C101             LD      1(P1)            ; BY SUBTRACTING
2179 0BF1 FAEF             CAD     LO(P2)
2180 0BF3 C100             LD      0(P1)
2181 0BF5 FAEE             CAD     HI(P2)           ;IS TEXT LINE # >= LINE #?
2182 0BF7 9407             JP      $2               ;YES - STOP LOOKING
2183 0BF9 C102             LD      2(P1)            ;NO - TRY NEXT LINE IN TEXT
2184 0BFB 01               XAE
2185 0BFC C580             LD      @EREG(P1)        ; SKIP LENGTH OF LINE
2186 0BFE 90E8             JMP     $1
2187 0C00 31      $2:      XPAL    P1               ;SAVE ADDRESS OF FOUND LINE
2188 0C01 CAF3             ST      LABLLO(P2)       ; IN LABLHI AND LABLLO
2189 0C03 31               XPAL    P1
2190 0C04 35               XPAH    P1
2191 0C05 CAF2             ST      LABLHI(P2)
2192 0C07 35               XPAH    P1
2193 0C08 C2EF             LD      LO(P2)           ;WAS THERE AN EXACT MATCH?
2194 0C0A E101             XOR     1(P1)
2195 0C0C 9C07             JNZ     $3
2196 0C0E C2EE             LD      HI(P2)
2197 0C10 E100             XOR     0(P1)
2198 0C12 9C01             JNZ     $3               ;NO - FLAG THE ADDRESS
2199 0C14 3F               XPPC    P3               ;YES - RETURN NORMALLY
2200 0C15 C2F2    $3:      LD      LABLHI(P2)       ;SET SIGN BIT OF HIGH PART
2201 0C17 DC80             ORI     080              ; OF ADDRESS TO INDICATE
2202 0C19 CAF2             ST      LABLHI(P2)       ; INEXACT MATCH OF LINE #'S
2203 0C1B 3F               XPPC    P3
2204
```

```
2205                    .PAGE   ' I.  L.  MACROS'
2206
2207                ;************************************
2208                ;*           I.  L.  MACROS          *
2209                ;************************************
2210
2211                    .LOCAL
2212
2213    2000    $TSTBIT =       TSTBIT*256
2214    8000    $CALBIT =       CALBIT*256
2215    4000    $JMPBIT =       JMPBIT*256
2216
2217                    .MACRO  TST,FAIL,A,B
2218                    .DBYTE  FAIL & OFFF ! $TSTBIT
2219                    .IF     #=2
2220                    .BYTE   'A'!080
2221                    .ELSE
2222                    .ASCII  'A'
2223                    .BYTE   'B'!080
2224                    .ENDIF
2225                    .ENDM
2226
2227                    .MACRO  TSTCR,FAIL
2228                    .DBYTE  FAIL & OFFF ! $TSTBIT
2229                    .BYTE   0D!080
2230                    .ENDM
2231
2232                    .MACRO  TSTV,FAIL
2233                    .ADDR   TSTVAR & OFFF
2234                    .DBYTE  FAIL
2235                    .ENDM
2236
2237                    .MACRO  TSTN,FAIL
2238                    .ADDR   TSTNUM & OFFF
2239                    .DBYTE  FAIL
2240                    .ENDM
2241
2242                    .MACRO  JUMP,ADR
2243                    .DBYTE  ADR & OFFF ! $JMPBIT
2244                    .ENDM
2245
2246                    .MACRO  CALL,ADR
2247                    .DBYTE  ADR & OFFF ! $CALBIT
2248                    .ENDM
2249
2250                    .MACRO  DO
2251                    .MLOC   I
2252                    .SET    I,1
2253                    .DO     #
2254                    .ADDR   #I & OFFF
2255                    .SET    I,I+1
2256                    .ENDDO
2257                    .ENDM
```

```
2258                        .PAGE   ' I. L. TABLE'
2259
2260                ;**************************************
2261                ;*              I. L. TABLE           *
2262                ;**************************************
2263
2264 0C1C          START:  DO      NLINE
2265 0C1E          PROMPT: DO      GETL
2266 0C20                  TSTCR   PRMPT1
2267 0C23                  JUMP    PROMPT
2268 0C25          PRMPT1: TSTN    LIST
2269 0C29                  DO      FNDPGE,XCHGP1,POPAE,FNDLBL,INSRT
2270 0C33                  JUMP    PROMPT
2271
2272 0C35          LIST:   TST     RUN,'LIS','T'
2273 0C3B                  DO      FNDPGE
2274 0C3D                  TSTN    LIST1
2275 0C41                  DO      POPAE,FNDLBL
2276 0C45                  JUMP    LIST2
2277 0C47          LIST1:  DO      CHPAGE
2278 0C49          LIST2:  DO      LST
2279 0C4B          LIST3:  CALL    PRNUM
2280 0C4D                  DO      LST3
2281 0C4F                  JUMP    START
2282
2283 0C51          RUN:    TST     CLR,'RU','N'
2284 0C56                  DO      DONE
2285 0C58          BEGIN:  DO      FNDPGE,CHPAGE,STRT,NXT
2286
2287 0C60          CLR:    TST     NEW,'CLEA','R'
2288 0C67                  DO      DONE,CLEAR,NXT
2289
2290 0C6D          NEW:    TST     STMT,'NE','W'
2291 0C72                  TSTN    DFAULT
2292 0C76                  JUMP    NEW1
2293 0C78          DFAULT: DO      LIT1
2294 0C7A          NEW1:   DO      DONE,POPAE,NUPAGE,FNDPGE,NEWPGM,NXT
2295
2296 0C86          STMT:   TST     LET,'LE','T'
2297 0C8B          LET:    TSTV    AT
2298 0C8F                  TST     SYNTAX,'='
2299 0C92                  CALL    RELEXP
2300 0C94                  DO      STORE,DONE,NXT
2301 0C9A          AT:     TST     IF,'@'
2302 0C9D                  CALL    FACTOR
2303 0C9F                  TST     SYNTAX,'='
2304 0CA2                  CALL    RELEXP
2305 0CA4                  DO      MOVE,DONE,NXT
2306
2307 0CAA          IF:     TST     UNT,'I','F'
2308 0CAE                  CALL    RELEXP
2309 0CB0                  TST     IF1,'THE','N'
2310 0CB6          IF1:    DO      POPAE,CMPR
2311 0CBA                  JUMP    STMT
```

```
2312
2313 0CBC          UNT:    TST     DO,'UNTI','L'
2314 0CC3                  DO      CKMODE
2315 0CC5                  CALL    RELEXP
2316 0CC7                  DO      DONE,POPAE,UNTIL,DETPGE,NXT
2317
2318 0CD1          DO:     TST     GOTO,'D','O'
2319 0CD5                  DO      CKMODE,DONE,SAVEDO,NXT
2320
2321 0CDD          GOTO:   TST     RETURN,'G','O'
2322 0CE1                  TST     GOSUB,'T','O'
2323 0CE5                  CALL    RELEXP
2324 0CE7                  DO      DONE
2325 0CE9                  JUMP    GO1
2326 0CEB          GOSUB:  TST     SYNTAX,'SU','B'
2327 0CF0                  CALL    RELEXP
2328 0CF2                  DO      DONE,SAV
2329 0CF6          GO1:    DO      FNDPGE,POPAE,FNDLBL,XFER,NXT
2330
2331 0D00          RETURN: TST     NEXT,'RETUR','N'
2332 0D08                  DO      DONE,RSTR,DETPGE,NXT
2333
2334 0D10          NEXT:   TST     FOR,'NEX','T'
2335 0D16                  DO      CKMODE
2336 0D18                  TSTV    SYNTAX
2337 0D1C                  DO      DONE,NEXTV
2338 0D20                  CALL    GTROP
2339 0D22                  DO      POPAE,NEXTV1,DETPGE,NXT
2340
2341 0D2A          FOR:    TST     STAT,'FO','R'
2342 0D2F                  DO      CKMODE
2343 0D31                  TSTV    SYNTAX
2344 0D35                  TST     SYNTAX,'='
2345 0D38                  CALL    RELEXP
2346 0D3A                  TST     SYNTAX,'T','O'
2347 0D3E                  CALL    RELEXP
2348 0D40                  TST     FOR1,'STE','P'
2349 0D46                  CALL    RELEXP
2350 0D48                  JUMP    FOR2
2351 0D4A          FOR1:   DO      LIT1
2352 0D4C          FOR2:   DO      DONE,SAVFOR,STORE,NXT
2353
2354 0D54          STAT:   TST     PGE,'STA','T'
2355 0D5A                  TST     SYNTAX,'='
2356 0D5D                  CALL    RELEXP
2357 0D5F                  DO      POPAE,MOVESR
2358 0D63                  DO      DONE,NXT
2359
2360 0D67          PGE:    TST     DOLLAR,'PAG','E'
2361 0D6D                  TST     SYNTAX,'='
2362 0D70                  CALL    RELEXP
2363 0D72                  DO      DONE,POPAE,NUPAGE,FNDPGE,CHPAGE,NXT
2364
2365 0D7E          DOLLAR: TST     PRINT,'$'
```

```
2366 0D81                CALL    FACTOR
2367 0D83                TST     SYNTAX,'='
2368 0D86                TST     DOLR1,'"'
2369 0D89                DO      POPAE,PUTSTR
2370 0D8D                JUMP    DOLR2
2371 0D8F        DOLR1:  TST     SYNTAX,'$'
2372 0D92                CALL    FACTOR
2373 0D94                DO      XCHGP1,MOVSTR,XCHGP1
2374 0D9A        DOLR2:  DO      DONE,NXT
2375
2376 0D9E        PRINT:  TST     INPUT,'P','R'
2377 0DA2                TST     PR1,'IN','T'
2378 0DA7        PR1:    TST     PR2,'"'
2379 0DAA                DO      PRS
2380 0DAC                JUMP    COMMA
2381 0DAE        PR2:    TST     PR3,'$'
2382 0DB1                CALL    FACTOR
2383 0DB3                DO      XCHGP1,POPAE,PSTRNG,XCHGP1
2384 0DBB                JUMP    COMMA
2385 0DBD        PR3:    CALL    RELEXP
2386 0DBF                CALL    PRNUM
2387 0DC1        COMMA:  TST     PR4,','
2388 0DC4                JUMP    PR1
2389 0DC6        PR4:    TST     PR5,';'
2390 0DC9                JUMP    PR6
2391 0DCB        PR5:    DO      NLINE
2392 0DCD        PR6:    DO      DONE,NXT
2393
2394 0DD1        INPUT:  TST     END,'INPU','T'
2395 0DD8                DO      CKMODE
2396 0DDA                TSTV    IN2
2397 0DDE                DO      XCHGP1,GETL
2398 0DE2        IN1:    CALL    RELEXP
2399 0DE4                DO      STORE,XCHGP1
2400 0DE8                TST     IN3,','
2401 0DEB                TSTV    SYNTAX
2402 0DEF                DO      XCHGP1
2403 0DF1                TST     SYNTAX,','
2404 0DF4                JUMP    IN1
2405 0DF6        IN2:    TST     SYNTAX,'$'
2406 0DF9                CALL    FACTOR
2407 0DFB                DO      XCHGP1,GETL,POPAE,ISTRNG,XCHGP1
2408 0E05        IN3:    DO      DONE,NXT
2409
2410 0E09        END:    TST     ML,'EN','D'
2411 0E0E                DO      DONE,BREAK
2412
2413 0E12        ML:     TST     REM,'LIN','K'
2414 0E18                CALL    RELEXP
2415 0E1A                DO      DONE,XCHGP1,POPAE,CALLML,XCHGP1,NXT
2416
2417 0E26        REM:    TST     SYNTAX,'RE','M'
2418 0E2B                DO      IGNORE,NXT
2419
```

```
2420 0E2F          SYNTAX: DO      ERR
2421 0E31          ERRNUM: CALL    PRNUM
2422 0E33                  DO      FIN
2423
2424               ; NOTE: EACH RELATIONAL OPERATOR (EQ, LEQ, ETC.) DOES AN
2425               ; AUTOMATIC 'RTN' (THIS SAVES VALUABLE BYTES AND TIME)
2426
2427 0E35          RELEXP: CALL    EXPR
2428 0E37                  TST     REL1,'='
2429 0E3A                  CALL    EXPR
2430 0E3C                  DO      EQ
2431 0E3E          REL1:   TST     REL4,'<'
2432 0E41                  TST     REL2,'='
2433 0E44                  CALL    EXPR
2434 0E46                  DO      LEQ
2435 0E48          REL2:   TST     REL3,'>'
2436 0E4B                  CALL    EXPR
2437 0E4D                  DO      NEQ
2438 0E4F          REL3:   CALL    EXPR
2439 0E51                  DO      LSS
2440 0E53          REL4:   TST     RETEXP,'>'
2441 0E56                  TST     REL5,'='
2442 0E59                  CALL    EXPR
2443 0E5B                  DO      GEQ
 444 0E5D          REL5:   CALL    EXPR
2445 0E5F          GTROP:  DO      GTR
2446
2447 0E61          EXPR:   TST     EX1,'-'
2448 0E64                  CALL    TERM
2449 0E66                  DO      NEG
2450 0E68                  JUMP    EX3
2451 0E6A          EX1:    TST     EX2,'+'
2452 0E6D          EX2:    CALL    TERM
2453 0E6F          EX3:    TST     EX4,'+'
2454 0E72                  CALL    TERM
2455 0E74                  DO      ADD
2456 0E76                  JUMP    EX3
2457 0E78          EX4:    TST     EX5,'-'
2458 0E7B                  CALL    TERM
2459 0E7D                  DO      SUB
2460 0E7F                  JUMP    EX3
2461 0E81          EX5:    TST     RETEXP,'O','R'
2462 0E85                  CALL    TERM
2463 0E87                  DO      OROP
2464 0E89                  JUMP    EX3
2465 0E8B          RETEXP: DO      RTN
2466
2467 0E8D          TERM:   CALL    FACTOR
2468 0E8F          T1:     TST     T2,'*'
2469 0E92                  CALL    FACTOR
 470 0E94                  DO      MUL
 471 0E96                  JUMP    T1
2472 0E98          T2:     TST     T3,'/'
2473 0E9B                  CALL    FACTOR
```

```
2474 0E9D                DO      DIV
2475 0E9F                JUMP    T1
2476 0EA1        T3:     TST     RETEXP,'AN','D'
2477 0EA6                CALL    FACTOR
2478 0EA8                DO      ANDOP
2479 0EAA                JUMP    T1
2480
2481 0EAC        FACTOR: TSTV    F1
2482 0EB0                DO      IND,RTN
2483 0EB4        F1:     TSTN    F2
2484 0EB8                DO      RTN
2485 0EBA        F2:     TST     F3,'#'
2486 0EBD                DO      HEX,RTN
2487 0EC1        F3:     TST     F4,'('
2488 0EC4                CALL    RELEXP
2489 0EC6                TST     SYNTAX,')'
2490 0EC9                DO      RTN
2491 0ECB        F4:     TST     F5,'@'
2492 0ECE                CALL    FACTOR
2493 0ED0                DO      EVAL,RTN
2494 0ED4        F5:     TST     F6,'NO','T'
2495 0ED9                CALL    FACTOR
2496 0EDB                DO      NOTOP,RTN
2497 0EDF        F6:     TST     F7,'STA','T'
2498 0EE5                DO      STATUS,RTN
2499 0EE9        F7:     TST     F8,'TO','P'
2500 0EEE                DO      FNDPGE,TOP,RTN
2501 0EF4        F8:     TST     F9,'MO','D'
2502 0EF9                CALL    DOUBLE
2503 0EFB                DO      DIV,MODULO,RTN
2504 0F01        F9:     TST     F10,'RN','D'
2505 0F06                CALL    DOUBLE
2506 0F08                DO      RANDOM,SUB,ADD,DIV,MODULO,ADD,RTN
2507 0F16        F10:    TST     SYNTAX,'PAG','E'
2508 0F1C                DO      PUTPGE,RTN
2509
2510 0F20        DOUBLE: TST     SYNTAX,'('
2511 0F23                CALL    RELEXP
2512 0F25                TST     SYNTAX,','
2513 0F28                CALL    RELEXP
2514 0F2A                TST     SYNTAX,')'
2515 0F2D                DO      RTN
2516
2517 0F2F        PRNUM:  DO      XCHGP1,PRN
2518 0F33        PRNUM1: DO      DIV,PRN1,XCHGP1,RTN
```

```
2519                            .PAGE   'ERROR MESSAGES'
2520
2521                    ;***************************************
2522                    ;*            ERROR MESSAGES           *
2523                    ;***************************************
2524
2525                            .MACRO  MESSAGE,A,B
2526                            .ASCII  'A'
2527                            .BYTE   'B'!080
2528                            .ENDM
2529
2530 0F3B                MESGS:  MESSAGE ' ERRO','R'     ; 1
2531 0F41                       MESSAGE 'ARE','A'       ; 2
2532 0F45                       MESSAGE 'STM','T'       ; 3
2533 0F49                       MESSAGE 'CHA','R'       ; 4
2534 0F4D                       MESSAGE 'SNT','X'       ; 5
2535 0F51                       MESSAGE 'VAL','U'       ; 6
2536 0F55                       MESSAGE 'END','"'       ; 7
2537 0F59                       MESSAGE 'NOG','O'       ; 8
2538 0F5D                       MESSAGE 'RTR','N'       ; 9
2539 0F61                       MESSAGE 'NES','T'       ; 10
2540 0F65                       MESSAGE 'NEX','T'       ; 11
2541 0F69                       MESSAGE 'FO','R'        ; 12
2542 0F6C                       MESSAGE 'DIV','O'       ; 13
2543 0F70                       MESSAGE 'BR','K'        ; 14
2544 0F73                       MESSAGE 'UNT','L'       ; 15
```

```
2545                        .PAGE   ' TELETYPE ROUTINES'
2546
2547                ;*************************************
2548                ;*     GET CHARACTER AND ECHO IT     *
2549                ;*************************************
2550                ;
2551                        .LOCAL
2552 0F77 C408      GECO:   LDI     8               ;SET COUNT = 8
2553 0F79 CAEB              ST      NUM(P2)
2554 0F7B 06                CSA                     ;SET READER RELAY
2555 0F7C DC02              ORI     2
2556 0F7E 07                CAS
2557 0F7F 06        $1:     CSA                     ;WAIT FOR START BIT
2558 0F80 D420              ANI     020
2559 0F82 9CFB              JNZ     $1              ;NOT FOUND
2560 0F84 C457 C3           LDI     87  195         ;DELAY 1/2 BIT TIME
2561 0F86 8F04 08           DLY     4    8
2562 0F88 06                CSA                     ;IS START BIT STILL THERE?
2563 0F89 D420              ANI     020
2564 0F8B 9CF2              JNZ     $1              ;NO
2565 0F8D 06                CSA                     ;SEND START BIT
2566 0F8E D4FD              ANI     %2              ;RESET READER RELAY
2567 0F90 DC01              ORI     1
2568 0F92 07                CAS
2569 0F93 C445 45$2:        LDI     133 69          ;DELAY 1 BIT TIME
2570 0F95 8F08 11           DLY     8    17
2571 0F97 06                CSA                     ;GET BIT (SENSEB)
2572 0F98 D420              ANI     020
2573 0F9A 9804              JZ      $3
2574 0F9C C401              LDI     1
2575 0F9E 9004              JMP     $4
2576 0FA0 C400      $3:     LDI     0
2577 0FA2 9C00              JNZ     $4
2578 0FA4 CAEA      $4:     ST      TEMP(P2)        ;SAVE BIT VALUE (0 OR 1)
2579 0FA6 1F                RRL                     ;ROTATE INTO LINK
2580 0FA7 01                XAE
2581 0FA8 1D                SRL                     ;SHIFT INTO CHARACTER
2582 0FA9 01                XAE                     ;RETURN CHAR TO E
2583 0FAA 06                CSA                     ;ECHO BIT TO OUTPUT
2584 0FAB DC01              ORI     1
2585 0FAD E2EA              XOR     TEMP(P2)
2586 0FAF 07                CAS
2587 0FB0 BAEB              DLD     NUM(P2)         ;DECREMENT BIT COUNT
2588 0FB2 9CDF              JNZ     $2              ;LOOP UNTIL 0
2589 0FB4 06                CSA                     ;SET STOP BIT
2590 0FB5 D4FE              ANI     0FE
2591 0FB7 07                CAS
2592 0FB8 8F08 11           DLY     8   17          ;DELAY APPROX.  1 BIT TIME
2593 0FBA 40                LDE                     ;AC HAS INPUT CHARACTER
2594 0FBB D47F              ANI     07F
2595 0FBD 01                XAE
2596 0FBE 40                LDE
2597 0FBF 3F                XPPC    P3              ;RETURN
2598 0FC0 90B5              JMP     GECO
```

```
2599
2600                ;****************************************
2601                ;*      PRINT CHARACTER AT TTY          *
2602                ;****************************************
2603
2604 0FC2 01       PUTC:   XAE
2605 0FC3 C4FF BB          LDI     255  187        ;DELAY ALMOST
2606 0FC5 8F17 2F          DLY     23    47        ; 3 BIT TIMES
2607 0FC7 06               CSA                     ;SET OUTPUT BIT TO LOGIC 0
2608 0FC8 DC01             ORI     1               ; FOR START BIT
2609 0FCA 07               CAS
2610 0FCB C409             LDI     9               ;INITIALIZE BIT COUNT
2611 0FCD CAE8             ST      TEMP3(P2)
2612 0FCF C48A 54  PUTC1:  LDI     138  84         ;DELAY 1 BIT TIME
2613 0FD1 8F08 11          DLY     8    17
2614 0FD3 BAE8             DLD     TEMP3(P2)       ;DECREMENT BIT COUNT.
2615 0FD5 9810             JZ      PUTC2
2616 0FD7 40               LDE                     ;PREPARE NEXT BIT
2617 0FD8 D401             ANI     1
2618 0FDA CAE9             ST      TEMP2(P2)
2619 0FDC 01               XAE                     ;SHIFT DATA RIGHT 1 BIT
2620 0FDD 1C               SR
2621 0FDE 01               XAE
2622 0FDF 06               CSA                     ;SET UP OUTPUT BIT
2623 0FE0 DC01             ORI     1
2624 0FE2 E2E9             XOR     TEMP2(P2)
2625 0FE4 07               CAS                     ;PUT BIT TO TTY
2626 0FE5 90E8             JMP     PUTC1
2627 0FE7 06       PUTC2:  CSA                     ;SET STOP BIT
2628 0FE8 D4FE             ANI     0FE
2629 0FEA 07               CAS
2630 0FEB 3F               XPPC    P3              ;RETURN
2631 0FEC 90D4             JMP     PUTC
2632      0000             .END    0
```

ADD	032D	AESTK	1050	ANDOP	05E8	AT	0C9A
BEGIN	0C58	BREAK	0280	CALBIT	0080	CALLML	0967
CHEAT	007C	CHEAT1	00C0	CHPAGE	0BBE	CHRNUM	FFE7
CK1	0641	CKMODE	063C	CLEAR	0051	CLEAR1	0056
CLR	0C60	CMP	055A	CMP1	05BA	CMP2	05C2
CMPR	05D1	COMMA	0DC1	DETPGE	0BC5	DFAULT	0C78
DIV	0408	DO	0CD1	DOLLAR	0D7E	DOLR1	0D8F
DOLR2	0D9A	DONE	0130	DONE1	013E	DONE2	013F
DOPTR	FFFF	DOSTAK	107A	DOUBLE	0F20	EO	014B
EOA	0108	E1	018D	E10	07CA	E11	0822
E12	0871	E12A	08E5	E13	0914	E14	0954
E15	0980	E16	09D0	E16A	0A32	E17	0A4E
E18	0AAB	E19	0B08	E2	01C4	E3A	0282
E4	02D7	E5	0304	E6	0370	E6A	03CA
E8	0643	E8A	06DD	E8B	06AA	E9	0753
END	0E09	EQ	0544	EREG	FF80	ERR	021B
ERR1	021D	ERR2	021F	ERRNUM	0E31	EVAL	07EA
EX1	0E6A	EX2	0E6D	EX3	0E6F	EX4	0E78
EX5	0E81	EXECIL	0076	EXPR	0E61	F1	0EB4
F10	0F16	F2	0EBA	F3	0EC1	F4	0ECB
F5	0ED4	F6	0EDF	F7	0EE9	F8	0EF4
F9	0F01	FACTOR	0EAC	FAIL	05D9	FAILHI	FFEC
FAILLO	FFED	FALSE	05C0	FIN	02AA	FNDLBL	0BE2
FNDPGE	0B98	FOR	0D2A	FOR1	0D4A	FOR2	0D4C
FORPTR	FFFE	FORSTK	108A	GECO	0F77	GEQ	0558
GEQ1	05B6	GETL	0757	GO1	0CF6	GOSUB	0CEB
GOTO	0CDD	GTR	0554	GTR1	05AB	GTROP	0E5F
HEX	064C	HI	FFEE	HILINE	FFF7	IF	0CAA
IF1	0CB6	IGNORE	09B9	ILC1	00A8	ILCALL	009E
IN1	0DE2	IN2	0DF6	IN3	0E05	IND	052C
INPUT	0DD1	INSRT	0824	ISTRNG	0B26	JMPBIT	0040
LABLHI	FFF2	LABLLO	FFF3	LBUF	10D6	LEQ	0550
LEQ1	05A2	LET	0C8B	LIST	0C35	LIST1	0C47
LIST2	0C49	LIST3	0C4B	LISTNG	FFF5	LIT1	0A34
LO	FFEF	LOLINE	FFF8	LSS	054C	LSS1	059A
LST	02D9	LST2	02F7	LST3	0306	LST4	030C
LST5	031C	LSTK	FFFD	MESGS	0F3B	ML	0E12
MODULO	09C0	MOVE	0808	MOVESR	094D	MOVSTR	0B56
MUL	0372	NEG	035B	NEQ	0548	NEQ1	0591
NEW	0C6D	NEW1	0C7A	NEWPGM	0BD1	NEXT	0D10
NEXTV	0A89	NEXTV1	0AEE	NLINE	020D	NOJUMP	009B *
NOTOP	05F0	NUM	FFEB	NUPAGE	0B8D	NXT	0284
NXT1	02A1	OROP	05EC	P1	0001	P1HIGH	FFF0
P1LOW	FFF1	P2	0002	P3	0003	PAGE	FFF6
PCHIGH	FFFA	PCLOW	FFFB	PCSTAK	10A6	PCSTK	FFF9
PGE	0D67	PGM	1120	POPAE	0916	PR1	0DA7
PR2	0DAE	PR3	0DBD	PR4	0DC6	PR5	0DCB
PR6	0DCD	PRINT	0D9E	PRMPT1	0C25	PRN	018F
PRN1	01C6	PRNUM	0F2F	PRNUM1	0F33	PROMPT	0C1E
PRS	0176	PRS1	018B	PSTRNG	0B0A	PUTC	0FC2
PUTC1	0FCF	PUTC2	0FE7	PUTPGE	0B7B	PUTSTR	0B36
RANDOM	09D2	REL1	0E3E	REL2	0E48	REL3	0E4F
REL4	0E53	REL5	0E5D	RELEXP	0E35	REM	0E26
RETEXP	0E8B	RETURN	0D00	RNDF	FFE6	RNDX	FFE5

SC/MP ASSEMBLER REV-A
NIBL 12/17/76

RNDY	FFE4	RSTR	0143	RSTR1	014D	RSTR2	015F
RTN	00F6	RUN	0C51	RUNMOD	FFF4	SAV	010A
SAV1	0126	SAV2	012C	SAVEDO	0978	SAVFOR	0A46
SBRPTR	FFFC	SBRSTK	106A	SETZ	0585	START	0C1C
STAT	0D54	STATUS	0956	STMT	0C86	STORE	04C1
STRT	02C0	SUB	0344	SYNTAX	0E2F	T1	0E8F
T2	0E98	T3	0EA1	TEMP	FFEA	TEMP2	FFE9
TEMP3	FFE8	TERM	0E8D	TOP	0994	TST	00C2
TSTBIT	0020	TSTNUM	06AC	TSTVAR	04E1	UNT	0CBC
UNTIL	0928	VARS	101C	X0	00E7	X1	015D
X10	04DA	X11	0542	X12	058F	X12A	05E6
X12B	062F	X12C	0672	X13	06DB	X14	0755
X15	07B8	X16	07E8	X17	0820	X19	086F
X19A	08E9	X20	090D	X21	0952	X22	0992
X23	09CE	X24	0A30	X25	0A87	X26	0AEC
X27	0B34	X4	01C2	X5	0219	X5A	027E
X6	02D5	X6A	0302	X7	0342	X8	036E
X9	03E7	X9A	0431	X9B	049C	XCHGP1	0631
XFER	0169	XFER1	0171	ZZ0001	101C	ZZ0002	1120
ZZ0003	0FC1	ZZ0004	0FC1	ZZ0005	0FC1	ZZ0006	0FC1
ZZ0007	0F3B	ZZ0008	0F3B	ZZ0009	0FC1	ZZ000A	10D6
ZZ000B	0FC1	ZZ000C	101C	ZZ000D	0FC1	ZZ000E	0002
ZZ000F	0002	ZZ0010	0006	ZZ0011	0002	ZZ0012	0003
ZZ0013	0002	ZZ0014	0002	ZZ0015	0002	ZZ0016	0002
ZZ0017	0005	ZZ0018	0004	ZZ0019	0002	ZZ001A	0007
ZZ001B	0004	ZZ001C	0004	ZZ001D	0003	ZZ001E	0002
ZZ001F	0006	ZZ0020	0005	ZZ0021	0002	ZZ0022	0003
ZZ0023	0006	ZZ0024	0005	ZZ0025	0002	ZZ0026	0003
ZZ0027	0005	ZZ0028	0002	ZZ0029	0002	ZZ002A	0005
ZZ002B	0003	ZZ002C	0003	ZZ002D	0007	ZZ002E	0003
ZZ002F	0004	ZZ0030	0003	ZZ0031	0002	ZZ0032	0005
ZZ0033	0002	ZZ0034	0003	ZZ0035	0002	ZZ0036	0003
ZZ0037	0003	ZZ0038	0002	ZZ0039	0006	ZZ003A	0003
ZZ003B	0003	ZZ003C	0007	ZZ003D	0003	ZZ003E	0002
ZZ003F	0002	ZZ0040	0002	ZZ0041	0002	ZZ0042	0002
ZZ0043	0002	ZZ0044	0002	ZZ0045	0002	ZZ0046	0002
ZZ0047	0002	ZZ0048	0002	ZZ0049	0002	ZZ004A	0002
ZZ004B	0002	ZZ004C	0002	ZZ004D	0002	ZZ004E	0003
ZZ004F	0002	ZZ0050	0003	ZZ0051	0002	ZZ0052	0003
ZZ0053	0003	ZZ0054	0003	ZZ0055	0004	ZZ0056	0004
ZZ0057	0008	ZZ0058	0003	ZZ0059	0002	ZZ005A	0003
ZZ005B	0005	$0	002A	$0	0418	$0	0773
$0	099A	$0	0B95	$1	0043	$1	01BE
$1	0235	$1	038F	$1	043B	$1	05F2
$1	06DF	$1	0776	$1	083C	$1	0934
$1	0982	$1	09A0	$1	0A01	$1	0A50
$1	0A93	$1	0B16	$1	0B4A	$1	0BA7
$1	0BE8	$1	0F7F	$10	0AAD	$2	01F7
$2	0252	$2	03A0	$2	044C	$2	06FF
;	07B6 *	$2	0846	$2	09A7	$2	0AE1
$2	0B2C	$2	0C00	$2	0F93	$3	0259
$3	03CC	$3	04A4	$3	084E	$3	0AE3
$3	0C15	$3	0FA0	$4	03E9	$4	085B
$4	0FA4	$5	0862	$ABOR	06BE	$ADD	08E7

```
SC/MP ASSEMBLER REV-A
NIBL  12/17/76


$ADD1   0903    $CALB   8000    $CR     07DC    $DOWN   0889
$END    04BB    $END    06A4    $END    0B50    $ENT1   049E
$ENTE   0683    $ENTE   07BA    $ERR    0751    $EXIT   03FA
$FAIL   04F3    $JMPB   4000    $LETR   0674    $LOOP   002C
$LOOP   00D6    $LOOP   01FB    $LOOP   0239    $LOOP   03AE
$LOOP   0458    $LOOP   0664    $LOOP   06EF    $LOOP   09DD
$LOOP   0AD5    $LOOP   0B3C    $LOOP   0B6C    $LOOP   0BAE
$MAYB   0505    $MOVE   0873    $MSG    023F    $NEQ    00E9
$NOT    0620    $OK     0512    $OK     0680    $OR     060E
$POS    0433    $PRNT   01E7    $REDO   0940    $REDO   0AFA
$RET    06CE    $RUB    07D2    $SCAN   00C4    $SHIF   068A
$SHIF   070C    $SKIP   065C    $TSTB   2000    $UP     0899
$UP1    08A5    $UP2    08C6    $UP3    08C8    $UP4    08D8
$XH     07CC    $XU     07AA

NO ERROR LINES
SOURCE CHECKSUM = B6C1
INPUT FILE     2:NIBLSLOW.SRC
```

2.4 SHORT PROGRAM EXAMPLES

1000 ITERATION FOR/NEXT LOOP
EXECUTION TIME: 11 SECONDS

```
10 FOR X=1 TO 1000
20 NEXT X
30 END
```

1000 ITERATION CONDITIONAL GOTO LOOP
EXECUTION TIME: 63 SECONDS

```
10 X=0
20 X=X+1
30 IF X<1000 GOTO 20
40 END
```

1000 ITERATION INTEGER ARITHMETIC FUNCTIONS
EXECUTION TIME: 58 SECONDS

```
10 FOR X=1 TO 1000
20 Y=X/2*3+4-5
30 NEXT X
40 END
```

1000 ITERATION INTEGER ARITHMETIC WITH A SUBROUTINE CALL
EXECUTION TIME: 86 SECONDS

```
10 FOR X=1 TO 1000
20 Y=X/2*3+4-5
25 GOSUB 100
30 NEXT X
40 END
100 RETURN
```

THIS PROGRAM INPUTS TWO DIGIT HEXADECIMAL NUMBERS AND STORES THEM IN SEQUENTIAL MEMORY LOCATIONS.

```
THE PROGRAM:                          COMMENTS:*

10 R=5000                             LOCATION IN MEMORY FOR TWO DIGIT
                                      NUMBER TO BE BUFFERED WHILE CONVERTING
20 INPUT U                            INPUT THE STARTING MEMORY LOCATION
40 INPUT $R                           INPUT TWO DIGIT CHARACTER STRING
50 LET X=@(R):GOSUB 1000              CALL THE CONVERSION ROUTINE 1ST DIGIT
55 A=X                                SET THE A VARIABLE TO FIRST DIGIT
60 LET X=@(R+1):GOSUB 1000           CALL THE CONVERSION ROUTINE 2ND DIGIT
65 B=X                                SET THE B VARIABLE TO 2ND DIGIT
70 C=(16*A)+B                         CONVERT THE 2 HEXT DIGITS TO DECIMAL
80 LET @(U)=C                         STORE THE DECIMAL NUMBER
90 U=U+1                              INCREMENT THE COUNTER
1000 IF X=65 X=58                     IF A, CONVERT TO 10+ASCII OFFSET (48)
1010 IF X=65 X=59                     IF B, CONVERT TO 11+ASCII OFFSET (48)
1020 IF X=67 X=60                     IF C, CONVERT TO 12+ASCII OFFSET (48)
1030 IF X=68 X=61                     IF D, CONVERT TO 13+ASCII OFFSET (48)
1040 IF X=69 X=62                     IF E, CONVERT TO 14+ASCII OFFSET (48)
1050 IF X=70 X=63                     IF F, CONVERT TO 15+ASCII OFFSET (48)
1060 X=X-48                           SUBTRACT ASCII OFFSET
1070 RETURN                           RETURN TO THE MAIN PROGRAM
```

THE SUBROUTINE RETURNS VALUES 0 THROUGH 15 TO THE MAIN PROGRAM. STATEMENT 70 CALCULATES THE DECIMAL EQUIVALENT TO THE HEXADECIMAL NUMBER. STATEMENT 80 USES THE INDIRECT OPERATOR TO STORE THE NUMBER IN MEMORY. THE NUMBER IS STORED AS A HEXADECIMAL OR BINARY NUMBER.

*THESE COMMENTS ARE NOT PART OF THE PROGRAM STATEMENT. IF THEY WERE, THE REM QUALIFIER WOULD HAVE TO BE INSERTED AND A COLON PLACED AFTER EACH STATEMENT. IN NIBL, COMMENTS REQUIRE LARGE AMOUNTS OF MEMORY FOR STORAGE. THIS FACT MUST BE KEPT IN MIND WHEN USING COMMENTS.

WHEN THE ABOVE LISTED PROGRAM IS IN MEMORY, THE PRINT TOP COMMAND RETURNS A VALUE OF 4667. THE FIRST MEMORY LOCATION AVAILABLE FOR STORAGE OF A PROGRAM IS 4382. THEREFORE THE PROGRAM ABOVE REQUIRES 285 BYTES FOR STORAGE. IF THE COMMENTS HAD BEEN INCLUDED, THE AMOUNT OF MEMORY REQUIRED WOULD HAVE BEEN OVER 1000 BYTES.

EXECUTING THE PROGRAM AND INPUTTING HEX NUMBERS

```
RUN
? 6000     LOCATION TO START PLACING NUMBERS
? AA
? BB
? CC
? DD
? EE
? FF
? 01
? 02
? 03
? 04
? 05
? 06
? 07
? 08
? 09
? (CONTROL C--TO INTERUPT EXECUTION OF PROGRAM)

BRK AT 40
```

PRINTING OUT A FEW LOCATIONS USING THE INDIRECT OPERATOR, CHECKING OUT THE RESULT OF THE ABOVE EXECUTION.

```
PRINT @6000
170                    I. E.  170 DECIMAL IS AA IN HEX
PRINT @6001
187                    I. E.  187 DECIMAL IS BB IN HEX
PRINT @6002
204                    I. E.  204 DECIMAL IS CC IN HEX
```

THIS PROGRAM OUTPUTS TWO DIGIT HEXADECIMAL NUMBERS FROM SEQUENTIAL MEMORY LOCATIONS:

```
 4 INPUT U                        U IS THE START OF MEMORY TO READ
 5 INPUT V                        V IS END OF MEMORY TO READ
 10 FOR Y=U TO V                  SET UP FOR/NEXT LOOP
 11 LET X=@(Y)                    READ OUT NUMBER
 60 E=X/16                        CONVERT TO HEXADECIMAL
 70 F=MOD(X,16)
 80 LET @5000=E+48                ADD IN ASCII OFFSETS
*81 LET @5001=F+48
*82 LET @5002=13                  PUT IN CARRIAGE RETURN (OD IN HEX)
*91 LET Z=@5000:GOSUB 1000        SET UP PRINTABLE FOR A TO F IST DIGIT
*92 LET @5000=Z                   PUT PRINTABLE IN PRINT BUFFER
*93 LET Z=@5001:GOSUB 1000        SET UP PRINTABLE FOR A TO F 2ND DIGIT
*94 LET @5001=Z                   PUT PRINTABLE IN PRINT BUFFER
*95 R=5000                        ADDRESS PRINT BUFFER
 100 PRINT Y,$R                   DO THE PRINT OF SGRING FROM BUFFER
 110 NEXT Y
 120 END
 1000 IF Z=58 Z=65                SET UP A PRINTABLE
 1010 IF Z=59 Z=66                SET UP B PRINTABLE
 1020 IF Z=60 Z=67                SET UP C PRINTABLE
 1030 IF Z=61 Z=68                SET UP D PRINTABLE
 1040 IF Z=62 Z=69                SET UP E PRINTABLE
 1050 IF Z=63 Z=70                SET UP F PRINTABLE
 1060 RETURN
```

* NOTE: USE OF SEQUENTIAL STATEMENT NUMBERS IS BAD PRACTICE. IT LEAVES NO ROOM FOR INSERTING MORE STATEMENTS.

EXECUTING THE PROGRAM

PRINTING OUT THE SAME LOCATIONS AS INPUTTED TO IN THE PREVIOUS PROGRAM. NOTE LOCATIONS 6016 TO 6020 WERE NOT WRITTEN TO.

```
RUN
? 6000     STARTING LOCATION TO READ FROM
? 6020     ENDING LOCATION TO READ FROM
6000 AA
6001 BB
6002 CC
6003 DD
6004 EE
6005 FF
6006 01
6007 02
6008 03
6009 04
6010 05
6011 06
6012 07
6013 08
6014 09
6015 00
6016 DD
6017 DD
6018 DD
6019 DD
6020 DD

BRK AT 120
```

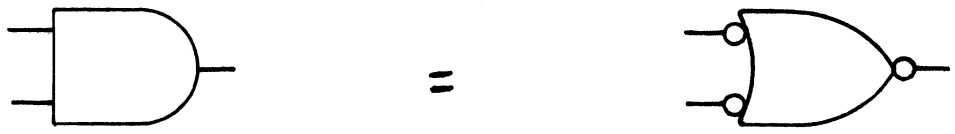
3. NIBBLER HARDWARE GUIDE

3.1 THEORY OF OPERATION

THE NIBBLER UTILIZES THE SC/MP II MICROPROCESSOR AND "NIBL" READ ONLY MEMORY INTEGRATED CIRCUITS. THESE COMPONENTS FORM THE BASIS FOR THE NIBBLER MICROCOMPUTER. THE FOLLOWING DISCUSSION PRESENTS THE RATIONALE BEHIND THE DESIGN USED FOR THE NIBBLER

3.1.1 LOGIC CONVENTION

THE LOGIC IS ARRANGED ACCORDING TO SIGNAL NAMES. ALMOST ALL SIGNALS ARE GIVEN SIGNAL NAMES. THE METHODOLOGY FOR SIGNAL NAMING IS DESCRIBED AS FOLLOWS: ALL POSITIVE LOGIC (TRUE WHEN HIGH) SIGNALS CONTAIN A P SUFFIX. ALL NEGATIVE LOGIC (TRUE WHEN LOW) SIGNALS CONTAIN AN N SUFFIX. SIGNALS AT THEIR SOURCE HAVE A 1 SUFFIX. FOR EXAMPLE ADDRESS LINE AD00 AT THE MICROPROCESSOR CHIP (THE SOURCE) IS INDICATED AS AD00-P1. THE "P" INDICATING TRUE WHEN HIGH AND THE "1" THE FIRST LEVEL OF LOGIC. AFTER THE SIGNAL IS BUFFERED BY A 81LS97, IT IS STILL TRUE WHEN HIGH, BUT A SECOND LEVEL SIGNAL. THEREFORE THE NAME IS CHANGED TO AD00-P2. IF AN INVERTING BUFFER WAS USED THE SIGNAL WOULD BE AD00-N1, I.E. THE FIRST LEVEL OF NEGATIVE LOGIC. MILITARY STANDARD 806B IS UTILIZED TO INDICATE GATE FUNCTION, FOR EXAMPLE, A1 IS A POSITIVE LOGIC "AND" GATE OR A NEGATIVE LOGIC "OR" GATE. THE GATE IS DRAWN AS IT IS UTILIZED IN THE CIRCUIT. MSI/LSI, WHERE CHIP SELECTS OR OUTPUTS ARE NEGATIVE WHEN TRUE, ARE INDICATED WITH BUBBLES.



AND NEG LOGIC OR

3.1.2 DISCUSSION OF SIGNALS

EACH OUTPUT OF THE SC/MP II WAS DESIGNED TO DRIVE ONE TTL LOAD. THEREFORE ALL OUTPUTS, WHEN UTILIZED, ARE BUFFERED USING EITHER A 81LS97 GATE OR A 74LS GATE. THE 81LS97 GATES ARE CAPABLE OF TRI-STATE OPERATION, HOWEVER THIS FEATURE WAS NOT UTILIZED. THE BUS PROVIDED HERE IS AN ACTIVE STATE SPLIT DIRECTION BUS. WE FELT THAT THIS BUS ARRANGEMENT WOULD BE EASY TO WORK WITH FOR THE NEW INITIATE INTO MICROCOMPUTING. THE DISCUSSION PRESENTED HERE CAN BE FOLLOWED BY USING THE BLOCK DIAGRAM AND THE DETAILED LOGIC PROVIDED WITH YOUR NIBBLER BOARD (SECTION 3.6).

THE SIGNAL NRST IS THE MASTER RESET FOR SC/MP. THE RC NETWORK CONNECTED TO GATE B2-18 PROVIDES A POWER-UP CLEAR TO THE MICROPROCESSOR CONFORMING TO THE 100 MSEC INITIALIZATION REQUIREMENT. IT WOULD BE A GOOD IDEA TO PROVIDE A SWITCH CLOSURE FROM GROUND TO PIN 49 OF THE EDGE CONNECTOR FOR RESET PURPOSES AFTER POWER UP.

THE FOLLOWING LIST OF STATUS SIGNALS, THEIR PRE-WIRED CONDITION AND A DESCRIPTION OF THEIR FUNCTION IS PROVIDED FOR YOUR REFERENCE.

SIGNAL MNEMONIC	FUNCTIONAL NAME	DESCRIPTION	NIBBLER CIRCUIT
CONT	CONTINUE INPUT	WHEN SET HIGH ENABLES NORMAL EXECUTION OF PROGRAM STORED IN EXTERNAL MEMORY. WHEN SET LOW, SC/MP OPERATION IS SUSPENDED AFTER COMPLETION OF CURRENT INSTRUCTION. WITHOUT LOSS OF INTERNAL STATUS.	PULLED HIGH WITH 10K TO 5V
NBREQ	BUS REQUEST INPUT/OUTPUT	ASSOCIATED WITH SC/MP INTERNAL ALLOCATION LOGIC FOR SYSTEM BUS. CAN BE USED AS A BUS REQUEST OUTPUT OR BUS BUSY INPUT. REQUIRES EXTERNAL LOAD RESISTORS TO VCC.	PULLED HIGH WITH 10K TO 5V
NENIN	ENABLE INPUT	ASSOCIATED WITH SC/MP INTERNAL ALLOCATION LOGIC FOR SYSTEM BUS. WHEN SET LOW, SC/MP IS GRANTED ACCESS TO SYSTEM BUSSES. WHEN SET HIGH, PLACES SYSTEM BUSSES IN HIGH IMPEDANCE (TRI-STATE) MODE.	JUMPERED LOW WITH JMP-A
NHOLD	INPUT/OUTPUT CYCLE EXTEND INPUT	WHEN SET LOW PRIOR TO TRAILING EDGE OF NRDS OR NWDS STROBE, STRETCHES STROBE TO EXTEND INPUT/OUTPUT CYCLE THAT IS, STROBE IS HELD LOW UNTIL NHOLD SIGNAL IS RETURNED HIGH	PULLED HIGH WITH 10K TO 5V

THE FOLLOWING LIST OF DYNAMIC SIGNALS, THEIR QUIESCENT STATE AND A DESCRIPTION IS PROVIDED FOR YOUR REFERENCE.

SIGNAL MNEMONIC	FUNCTIONAL NAME	DESCRIPTION	NIBBLER CIRCUIT
NRST	RESET INPUT	SET HIGH FOR NORMAL OPERATION. WHEN SET LOW, ABORTS IN PROCESS OPERATIONS. WHEN RETURNED HIGH, INTERNAL CONTROLS CIRCUIT ZEREOS ALL PROGRAMMER ACCESSABLE REGISTERS THEN THE FIRST INSTRUCTION IS FETCHED FROM MEMORY LOCATION 0001	DRIVEN BY B2-17 NORMAL HIGH
NADS	ADDRESS STROBE OUTPUT	ACTIVE LOW STROBE, WHILE LOW, INDICATES THAT VALID ADDRESS AND STATUS OUTPUT ARE PRESENT ON SYSTEM BUSSES.	HAS INTERNAL TERMINATION.
NRDS	READ STROBE OUTPUT	ACTIVE LOW STROBE. ON TRAILING EDGE DATA ARE INPUT TO SC/MP FROM 8-BIT BIDIRECTIONAL DATA BUS. HIGH IMPEDANCE (TRI-STATE) OUTPUT WHEN INPUT/OUTPUT CYCLE IS NOT IN PROGRESS.	REQUIRES 10K PULL UP
NWDS	WRITE STROBE OUTPUT	ACTIVE LOW STROBE, WHILE LOW, INDICATES THAT VALID OUTPUT DATA ARE PRESENT ON BIIRECTIONAL DATA BUS. HIGH IMPEDANCE OUTPUT WHEN INPUT/OUTPUT CYCLE NOT IN PROGRESS	REQUIRES 10K PULL UP
SENSE B	SENSE INPUT	USER DESIGNATED SENSE CONDITION INPUT SENSE CONDITION TESTING IS EFFECTED BY COPYING STATUS REGISTER TO ACCUMULATOR.	USED AS SERIAL DATA INPUT
FLAGS 0, 1	FLAG OUTPUTS	USER-DESIGNATED GENERAL PURPOSE FLAG OUTPUTS OFSTATUS REGISTER. UNDER PROGRAM CONTROL, FLAGS CAN BE SET AND RESET BY COPYING ACCUMULATOR TO STATUS REGISGER	USED AS SERIAL DATA OUTPUT
AD00-AD11	ADDRESS BIT 00 THROUGH ADDRESS BIT 11	TWELVE TRI-STATE ADDRESS OUTPUT LINES, SC/MP OUTPUTS 12 LEAST SIGNIFICANT ADDRESS BITS ON THIS BUS WHEN NADS STROBE IS LOW. ADDRESS BITS ARE THEN HELD UNTIL TRAILING EDGE OF READ (NRDS) OR WRITE (NWDS) STROBES. AFTER TRAILING EDGE OF NRDS OR NWDS STROBE, BUS IS SET TO TRI-STATE MODE UNTIL NEXT NADS STROBE.	BUFFERED BY C2 & B2

SIGNAL MNEMONIC	FUNCTIONAL NAME	DESCRIPTION	INPUT AT NRDS TIME	INPUT AT NWDS TIME	NIBBLER CIRCUIT
DB0	ADDRESS BIT 12	FOURTH MOST SIG NIFICANT BIT OF 16-BIT ADDRESS			
DB1	ADDRESS BIT 13	THIRD MOST SIG-NIFICANT BIT OF 16-BIT ADDRESS			
DB2	ADDRESS BIT 14	2ND MOST SIG-NIFICANT BIT OF 16-BIT ADDRESS.	INPUT DATA ARE EXPECT-ED ON THE EIGHT (DB0-DB7) LINES.	OUTPUT DATA ARE VALID ON THE EIGHT (DB0-DB7) LINES.	BUFFER-ED BY B6 & B8
DB3	ADDRESS BIT 15	MOST SIGNIFICAN BIT OF 16-BIT A DRESS.			
DB4	R-FLAG	WHEN HIGH, DATA INPUT CYCLE IS STARTING. WHEN LOW, DATA OUTPUT CYCLE IS START-ING			
DB5	1-FLAG	WHEN HIGH, FIRST BYTE OF INSTRUCT-ION IS BEING FETCHED.			
DB6	D-FLAG	WHEN HIGH, INDI-ATES DELAY CYCLE IS STARTING, IF SECOND BYTE OF DLY INSTRUCTION IS BEING FETCHED			
DB7	H-FLAG	WHEN HIGH, INDIC-ATES THAT HALT INSTRUCTION HAS BEEN EXECUTED, IN SOME SYSTEM CON-FIGURATIONS, THE H-FLAG OUTPUT IS LATCHED WITH THE CONTINUE INPUT TO PROVIDE A PROG-RAMMED HALT.	NOTE: THE DB0 THROUGH DB7 (AD12-HFLG) LINES ARE A HIGH IMPEDANCE (OPEN CIR-CUIT) LOAD WHEN SC/MP DOES NOT HAVE ACCESS TO THE INPUT/OUTPUT BUS		

THE REMAINING SC/MP SIGNALS SUCH AS NENOUT, FLG2, SENA, SIN, AND SOUT ARE NOT USED BUT ARE PROVIDED AT THE EDGE CONNECTOR FOR POSSIBLE FUTURE USE, THEY ARE NOT BUFFERED. THE BIDIRECTIONAL DATA BUS ARE BUFFERED OUT BY B6 AND BUFFERED IN BY B8. B6 IS ACTIVE AT ALL TIMES, B8 IS ACTIVE ONLY DURING NRDS TIME. THE ADDRESS LINES ARE ACTIVE AT ALL TIMES, THE CONTROL STROBES NADS, NRDS AND NWDS ARE BUFFERED BY B2. THE LEAST SIGNIFICANT DATA BITS DB00 THROUGH DB03 ARE LATCHED BY B5 TO BE UTIIZED AS ADDRESS BITS A12 THROUGH A15. NADS IS INVERTED BY A6 AND USED AS A STROBE TO LATCH DB00-DB03 IN B5. ADDRESS LINES A10, A11 AND A12 ARE DECODED BY A5 TO PROVIDE CHIP SELECTS AT 1K BOUNDARIES FOR THE ON-BOARD 6K AND OFF BOARD 2K MEMORY SPACE. THE NIBL ROM IS ORGANIZED IN 2K SECTIONS. THEREFORE, THE SELECTS FOR THE 0,1 BOUNDARIES ARE "OR'D" TOGETHER BY A2 TO PROVIDE A CHIP SELECT TO ROM 1. IN A SIMILAR MANNER THE 2,3 SELECTS ARE "OR'D" BY A2 TO PROVIDE A CHIP SELECT FOR ROM 2. CHIP SELECTS 4 AND 5 ARE USED BY THE ON BOARD 2K RAM. CHIP SELECTS 6 AND 7 ARE PROVIDED AT THE EDGE CONNECTOR PIN 27 FOR DESELECTING THE ON-BOARD DECODER FOR GREATER THAN 8K OF MEMORY SPACE. THIS DESELECT IS JUMPERED TO GROUND WITH JMP-C ON YOUR SHIPPED BOARD. THIS WILL ALLOW THE BOARD TO BE USED WITHOUT EXTERNAL MEMORY AND DECODING.

3.1.3 DETAILED DISCUSSION OF MEMORY ADDRESSING

THE MAIN COMPUTER BOARD CONTAINS THE NIBL INTERPRETER IN TWO 2316A ROMS. THE ROMS ARE ADDRESSED BY ADDRESS LINES AD00-P2 THROUGH AD10-P2. THE ADDRESS LINES SELECT 1 BYTE OUT OF A POSSIBLE 2048. THE 1 OF 2 2316'S IS SELECTED BY THE DECODE RESULTING FROM AN EIGHT LINE DECODER (74LS155). ADDRESS LINES AD10-P2, AD11-P2 AND AD12-P1 ARE USED TO DRIVE DECODER A5. THE DECODES RESULTING ARE ON 1K (1024) BOUNDARIES. THE TWO SELECTS (0-07FF), IN HEX, ARE "OR'D" TO PROVIDE A CHIP SELECT FOR THE FIRST NIBL ROM. THE SECOND TWO SELECTS (0800-0FFF) ARE "OR'D" TO PROVIDE A CHIP SELECT FOR THE SECOND NIBL ROM.

THE REMAINING FOUR DECODER OUTPUTS ARE USED AS CHIP SELECTS FOR 1K BYTE MODULES COMPOSED OF 8-21L02-4 MEMORY CHIPS. TWO 1K MODULES ARE PROVIDED ON THE MAIN COMPUTER BOARD. THE TWO MODULES ARE SELECTED BY DECODES 4 AND 5. THE TWO REMAINING DECODES 6 AND 7 ARE PROVIDED AS OUTPUTS ON THE EDGE CONNECTOR. THE CHIP SELECTS PROVIDED ALLOW AN ADDRESS SPACE OF 8K. THIS 8K SPACE CONSISTS OF 4K NIBL ROMS AND UP TO 4K OF RAM. EXPANSION OF THE MEMORY SPACE TO 32K INCLUSIVE OF THE INTERPRETER IS ALLOWED. OFF BOARD DECODING MUST BE PROVIDED FOR ADDITIONAL MEMORY EXPANSION. TO PREVENT WRAP-AROUND, I.E. ADDRESSING OF MAIN BOARD MEMORY AGAIN AT 8192, A DISABLE MUST BE PROVIDED TO THE EDGE CONNECTOR (PIN 27). THIS DISABLE CAN BE FORMED BY "ORING" THE ADDRESS LINES AD13 AND AD14. AD15 IS NOT REQUIRED AS THE ADDRESS SPACE OF NIBL DOES NOT INCLUDE THE UPPER 32K OF MEMORY. MEMORY ADDRESS SPACE IS SHOWN IN FIGURE 3.1.

MEMORY ADDRESS SPACE

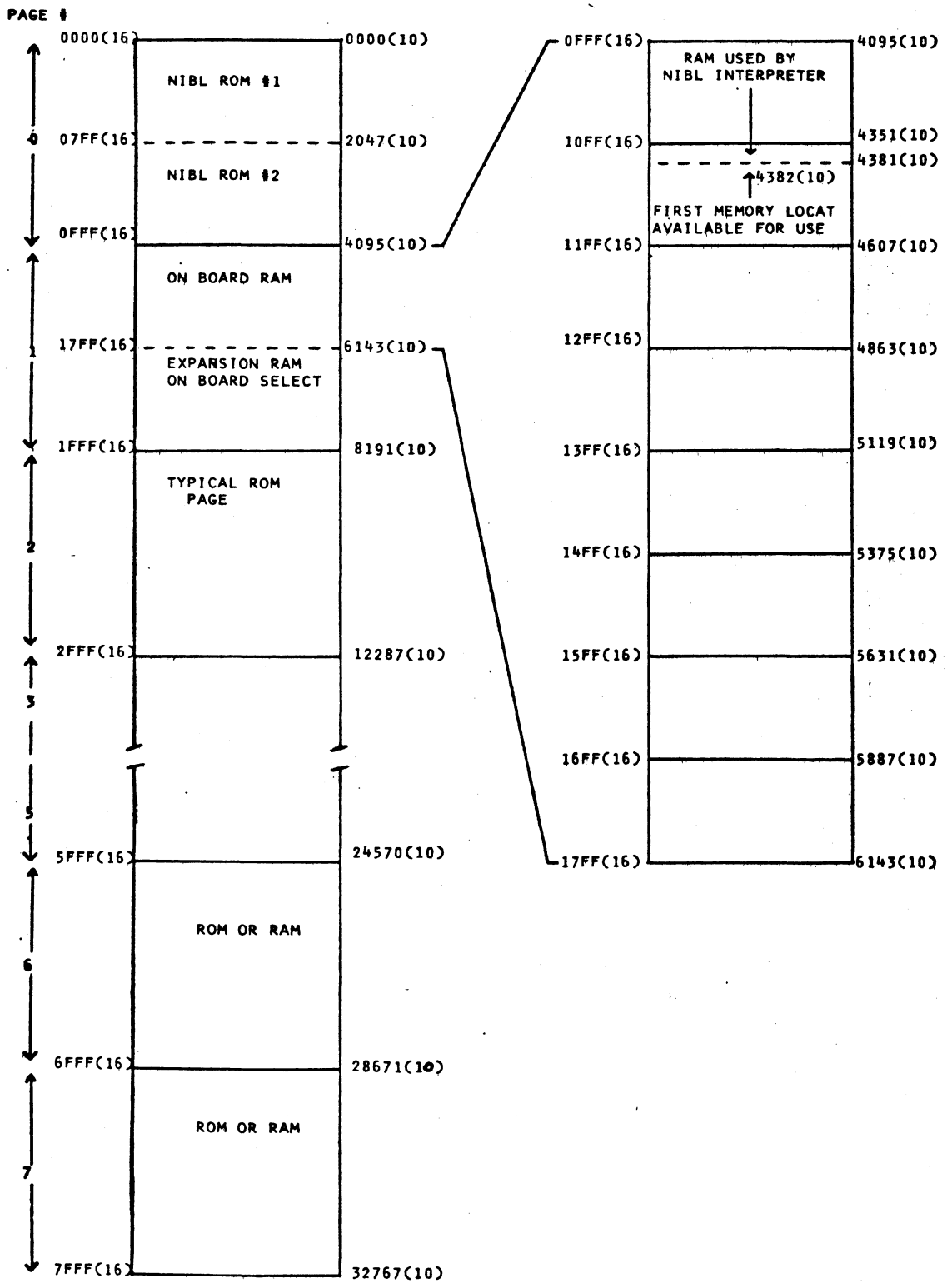


FIGURE 3.1 MEMORY ADDRESS SPACE

3.2 TYPICAL SYSTEM CONFIGURATION

THE SYSTEM SHOWN IN FIGURE 3.2 IS BUILT AROUND THE DIGI-KEY POWER SUPPLY AND INTERFACING BOARD WHICH IS AVAILABLE FOR PURCHASE WITH THE NIBBLER BOARD. THE POWER SUPPLY PROVIDES 5V AT 1A AND + & - 12V AT 100 MA. THE INTERFACE PROVIDES A TELETYPE AND RS232 I/O CAPABILITY FOR THE NIBBLER BOARD. THE INTERFACE BOARD ALSO PROVIDES A REED RELAY FOR CONTROLLING THE PAPER TAPE READER OF A TELETYPE. THE PAPER TAPE CAPABILITY OF A TELETYPE IS NECESSARY FOR PROGRAM STORAGE AND RETRIEVAL AT THIS TIME. A CRT TERMINAL CAN BE USED FOR PROGRAM DEVELOPMENT. THE TELETYPE CAN BE USED FOR HARDCOPY OUTPUT. THE PAPER TAPE READER AND PUNCH ARE USED FOR PROGRAM STORAGE AND RETRIEVAL. A DISABLE SWITCH CAN BE INSTALLED IN THE TELETYPE PAGE PRINTER LINE SO THAT THE CHARACTERS READ IN FROM THE PAPER TAPE ARE NOT PRINTED ON THE PAPER OUT. THIS ARRANGEMENT OF CRT AND TELETYPE MAKES A VERY EFFICIENT PROGRAM DEVELOPMENT SYSTEM.

THE LOGIC FOR A TELETYPE AND RS232 INTERFACE IS SHOWN IN FIGURE 3.2B. SEE SECTION 3.5.1 AND 3.5.2 FOR POWER AND TELETYPE INTERFACING DETAILS.

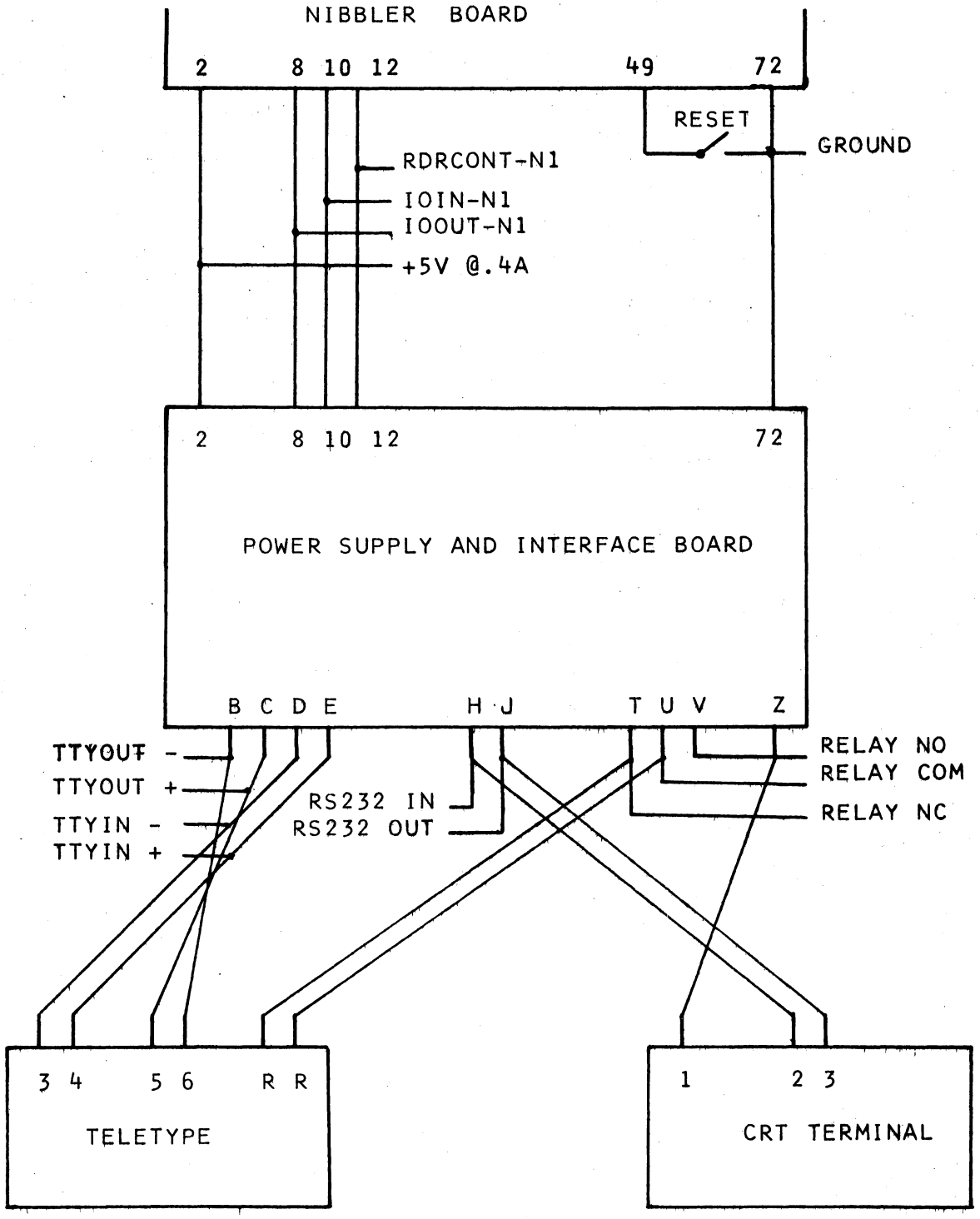


FIGURE 3.2A TYPICAL SYSTEM CONFIGURATION

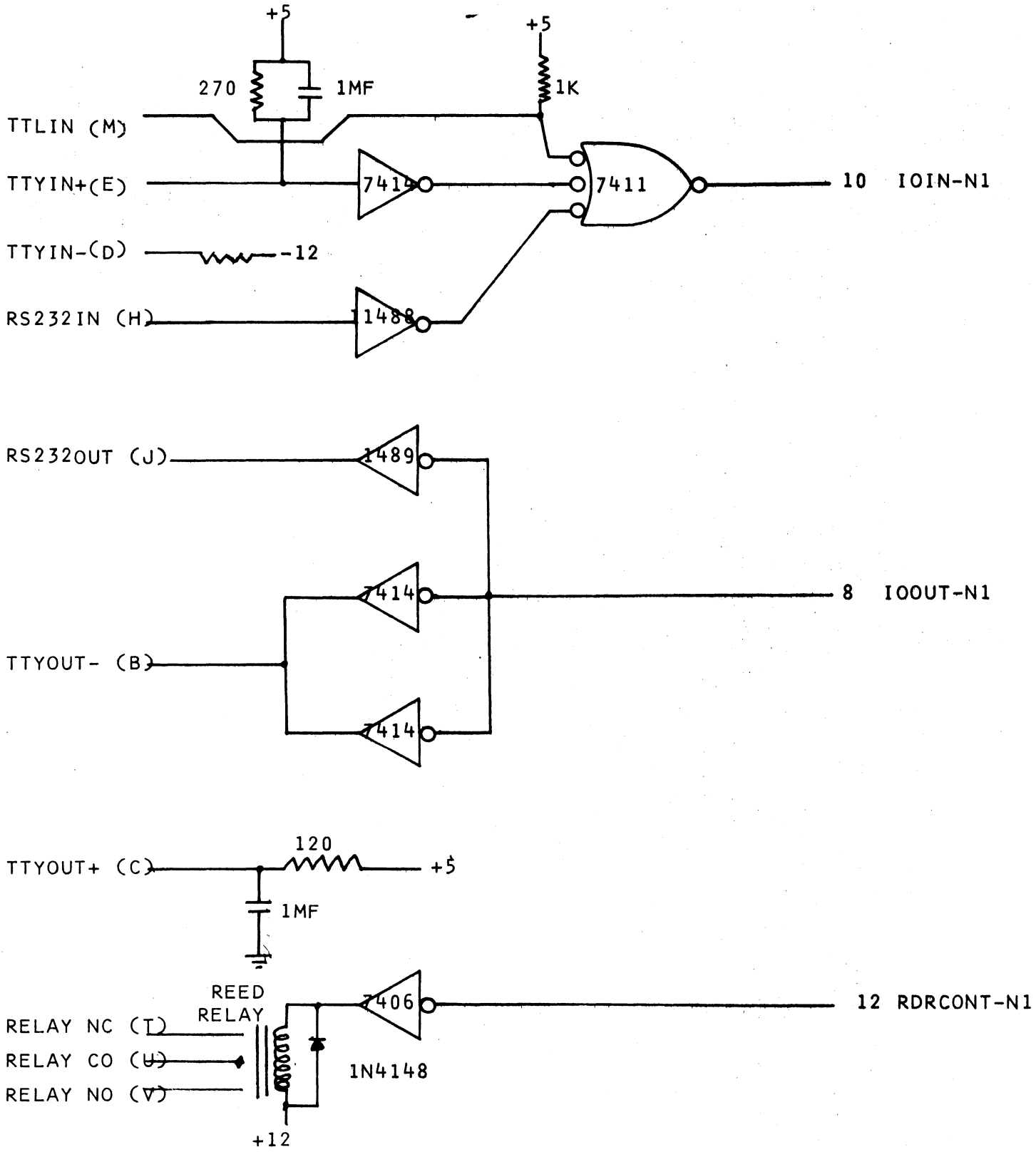


FIGURE 3.2B INTERFACE BOARD LOGIC

3.3 BACKPLANE WIRING

A BACKPLANE CAN BE PREPARED BY RUNNING BUS STRIPS DOWN THE +5V AND GROUND PINS FOR POWER AND WIREWRAPPING THE REST OF THE SIGNAL PINS TOGETHER. FIGURE 3.3 SHOWS A TYPICAL SYSTEM WIRED FOR USE WITH 6 CARDS. DIGI-KEY SELLS A CARD FILE CAPABLE OF HOLDING UP TO 14 CARDS. THE SIGNAL DESCRIPTIONS ARE GIVEN BY SHEET 2 OF THE LOGIC DIAGRAMS. THE SIGNALS ARE GROUPED BY FUNCTION WHERE POSSIBLE. THE DATA IN SIGNALS, DATA OUT SIGNALS, 12 LEAST SIGNIFICANT ADDRESS LINES, 4 MOST SIGNIFICANT ADDRESS LINES, AND I/O LINES ARE GROUPED TOGETHER.

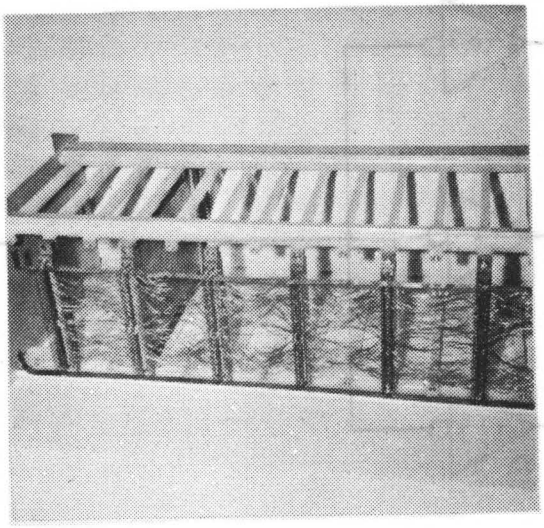


FIGURE 3.3 BACKPLANE WIRING WITH POWER SUPPLY DISTRIBUTION STRIPS

3.4 BOARD JUMPERING

1. JUMPER A IS PROVIDED SO THAT NENIN-NI MAY BE UTILIZED AS A CONTROL INPUT TO THE SC/MP. THE JUMPER WILL NORMALLY BE INSTALLED SO THAT NENIN-NI WILL BE GROUNDED. THE NIBBLER IS PROVIDED WITH THIS JUMPER.

2. JUMPER B IS PROVIDED TO ALLOW IOIN-N1 TO BE ACTIVE AS AN INPUT THE JUMPER MUST BE IN PLACE TO BE USED WITH THE POWER SUPPLY INTER-FACT BOARD.

3. JUMPER C IS PROVIDED TO SELECT THE ON BOARD MEMORY. THE BOARD IS PROVIDED WITH THIS JUMPER.

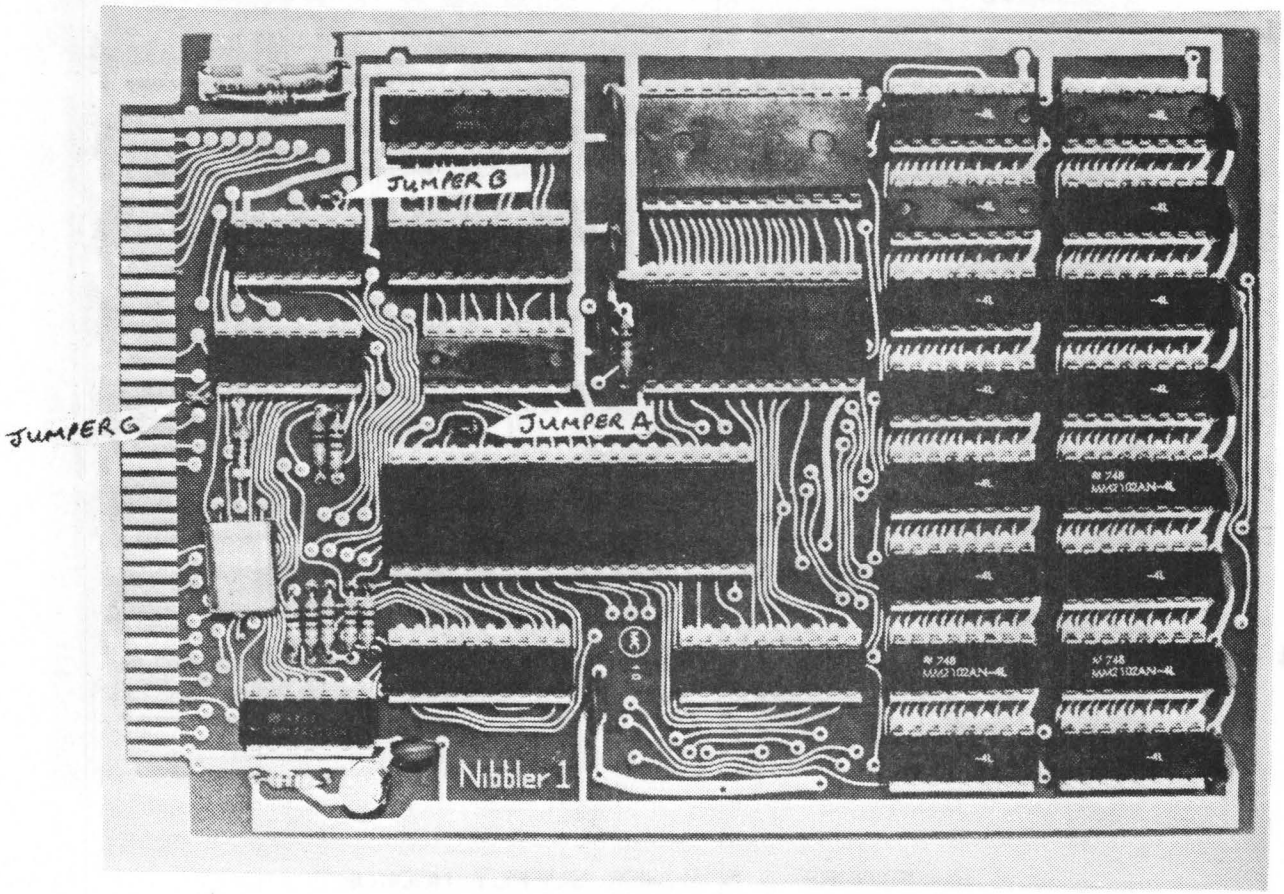


FIGURE 3.4 BOARD JUMPERING

3.5 INTERFACING

3.5.1 POWER INTERFACING

EDGE CONNECTOR PINS ARE PROVIDED ON THE PRINTED CIRCUIT BOARD FOR CONNECTION OF INPUT POWER. A SINGLE +5V SUPPLY IS ALL THAT IS REQUIRED EXCLUSIVE OF THE I/O INTERFACE. THE POWER MUST BE SUPPLIED AS SHOWN BELOW.

THE PINS ON THE REVERSE SIDE OF PIN 1(2) AND PIN 71(72) ARE PLATED THROUGH RESPECTIVELY.

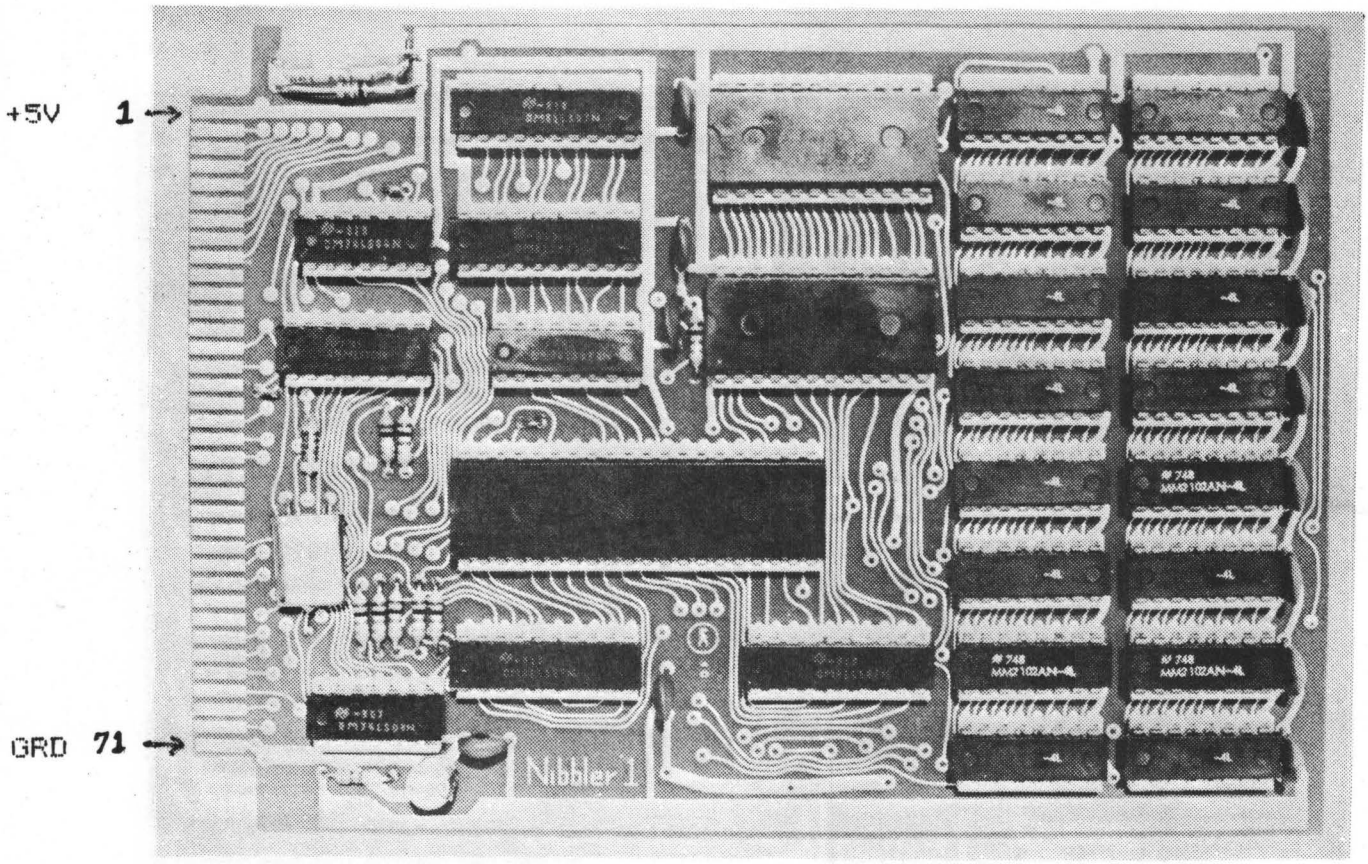


FIGURE 3.5 POWER SUPPLY HOOKUP

3.5.2 TELETYPE INTERFACING

TELETYPE SETUP AND SYSTEM CONNECTION

THE NIBBLER IS DESIGNED TO OPERATE WITH A STANDARD TELETYPE MODEL ASR3320/JC OR TU (WITH OR WITHOUT A PAPER TAPE READER/PUNCH) WITHOUT XON, XOFF, AND WITH THE AUTOMATIC ANSWERBACK OPTION DISABLED.

THE TTY MUST BE SET TO OPERATE IN THE FULL-DUPLEX MODE WITH A 20 MA CURRENT LOOP INTERFACE. INSTRUCTIONS FOR TTY SET UP AND CONNECTION TO THE NIBBLER ARE PROVIDED BELOW. FIGURE 3.6A IS A TOP VIEW OF THE TTY SHOWING THE LOCATION OF THE ASSEMBILES THAT ARE REFERRED TO IN THESE INSTRUCTIONS. FIGURES 3.6B AND 3.6C SHOW THE DETAILS OF THE TERMINAL STRIP AND CURRENT SOURCE RESISTOR. IN THESE FIGURES, THE DOTTED LINES INDICATE THE CONNECTIONS FOR HALF-DUPLEX AND 60 MA CURRENT LOOP OPERATION. THIS IS THE CONFIGURATION IN WHICH THE TTY IS NORMALLY SHIPPED FROM TELETYPE CORP. THE SOLID LINES INDICATE THE DESIRED CONECTIONS FOR FULL-DUPLEX AND 20 MA CURRENT LOOP OPERATION. ENSURE THAT POWER IS REMOVED FROM THE TTY BEFORE PERFORMING THE FOLLOWING STEPS.

1. TO SET TTY CURRENT SOURCE TO 20 MA, MOVE BLUE WIRE FROM TERMINAL 3 TO TERMINAL 4 OF THE CURRENT SOURCE RESISTOR.

2. TO SET RECEIVE CURRENT TO 20 MA, MOVE PURPLE WIRE FROM PIN 8 TO PIN 9 ON THE TERMINAL STRIP LOCATED AT REAR OF TTY.

3. TO CONFIGURE TTY FOR FULL-DUPLEX, MOVE WHITE-BLUE WIRE FROM PIN 4 TO PIN 5 ON THE TERMINAL STRIP, AND MOVE BROWN-YELLOW WIRE FROM PIN 3 TO PIN 5.

4. TO DISABLE THE AUTO-ANSWERBACK OPTION, LIFT THE PRINT STATION PAPER COVER AND LOCATE THE CAVITY BEHIND THE KEYBOARD. DIRECTLY BENEATH THE CARRIAGE IS A SET OF NINE CODEBARS. AT THE FRONT OF THIS ASSEMBLY IS A TIE-BAR. (SEE FIGURE 3.7). THE AUTO-ANSWERBACK IS DISABLED BY PLACING A CLIP OVER THE TIE-BAR SO THAT THE THIRD SLOT FROM THE RIGHT IS COVERED. ON SOME MODELS, ONE OF THESE COPPER-COLORED CLIPS ALREADY MAY BE PLACED OVER THE SECOND SLOT; IF SO MOVE IT TO THE THIRD SLOT. IF NO CLIP IS PROVIDED, IT CAN BE OBTAINED FROM YOUR LOCAL TELETYPE DEALER.

5. CONNECT TTYOUT (+) FROM 22 PIN CONN "B" TO PIN 6 ON THE TTY TERMINAL STRIP.

6. CONNECT TTYOUT (-) FROM 22 PIN CONN "C" TO PIN 7 ON THE TTY TERMINAL STRIP

7. CONNECT TTYIN (+) FROM 22 PIN CONN"E" TO PIN 4 ON THE TTY TERMINAL STRIP.

8. CONNECT TTYIN (-) FROM 22 PIN CONN"D" TO PIN 3 ON THE TTY TERMINAL STRIP.

NOTE:

CABLE LENGTH FROM THE TTY TO THE NIBBLER PS/I CARD SHOULD NOT EXCEED 12 FEET. RECOMMENDED CABLE TYPE IS STANDARD TWISTED PAIR, 22 AWG.

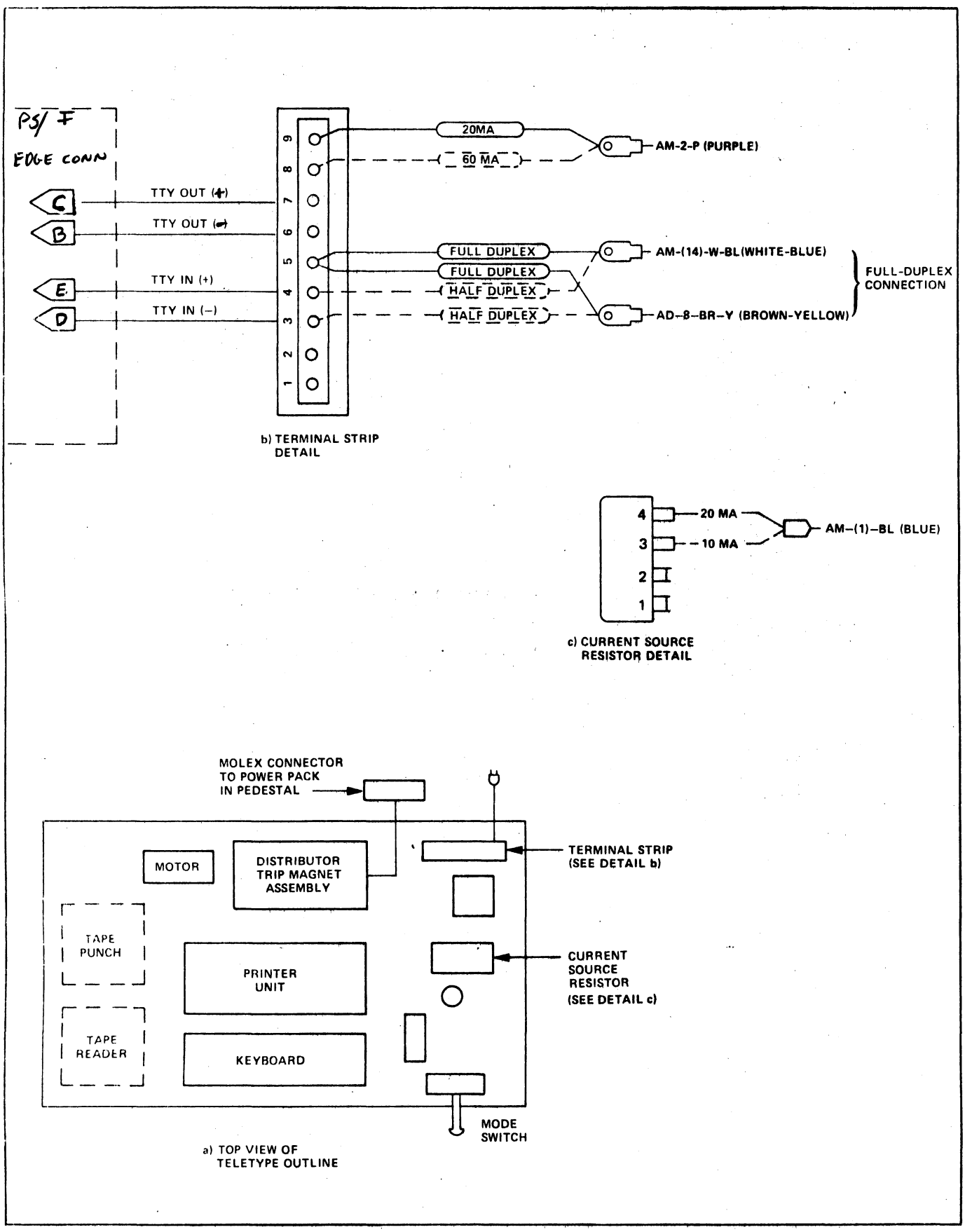


FIGURE 3.6 TELETYPE CONNECTIONS

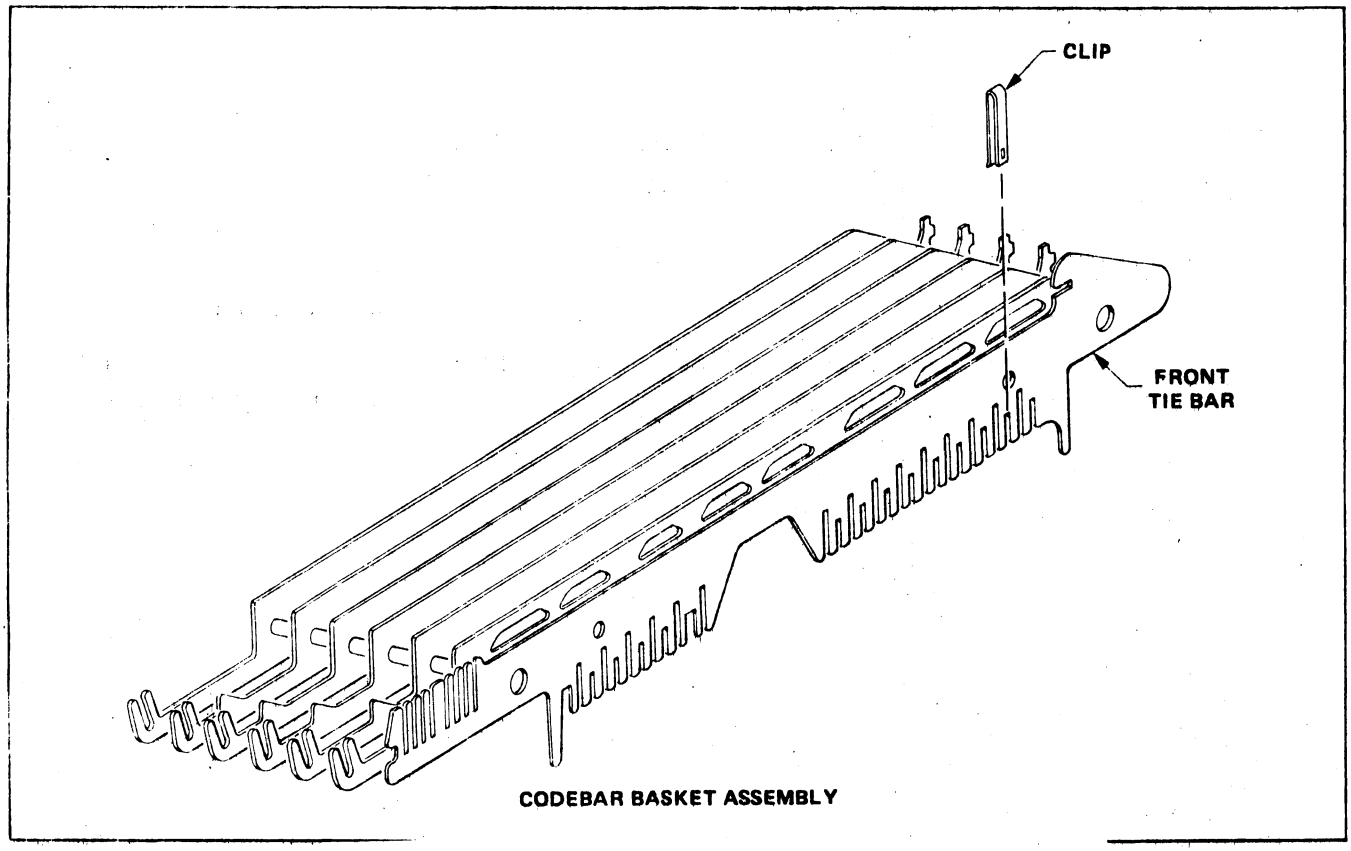


FIGURE 3.7 DISABLING TTY AUTO-ANSWERBACK

3.5.3 A SAMPLE I/O APPLICATION DISCUSSION

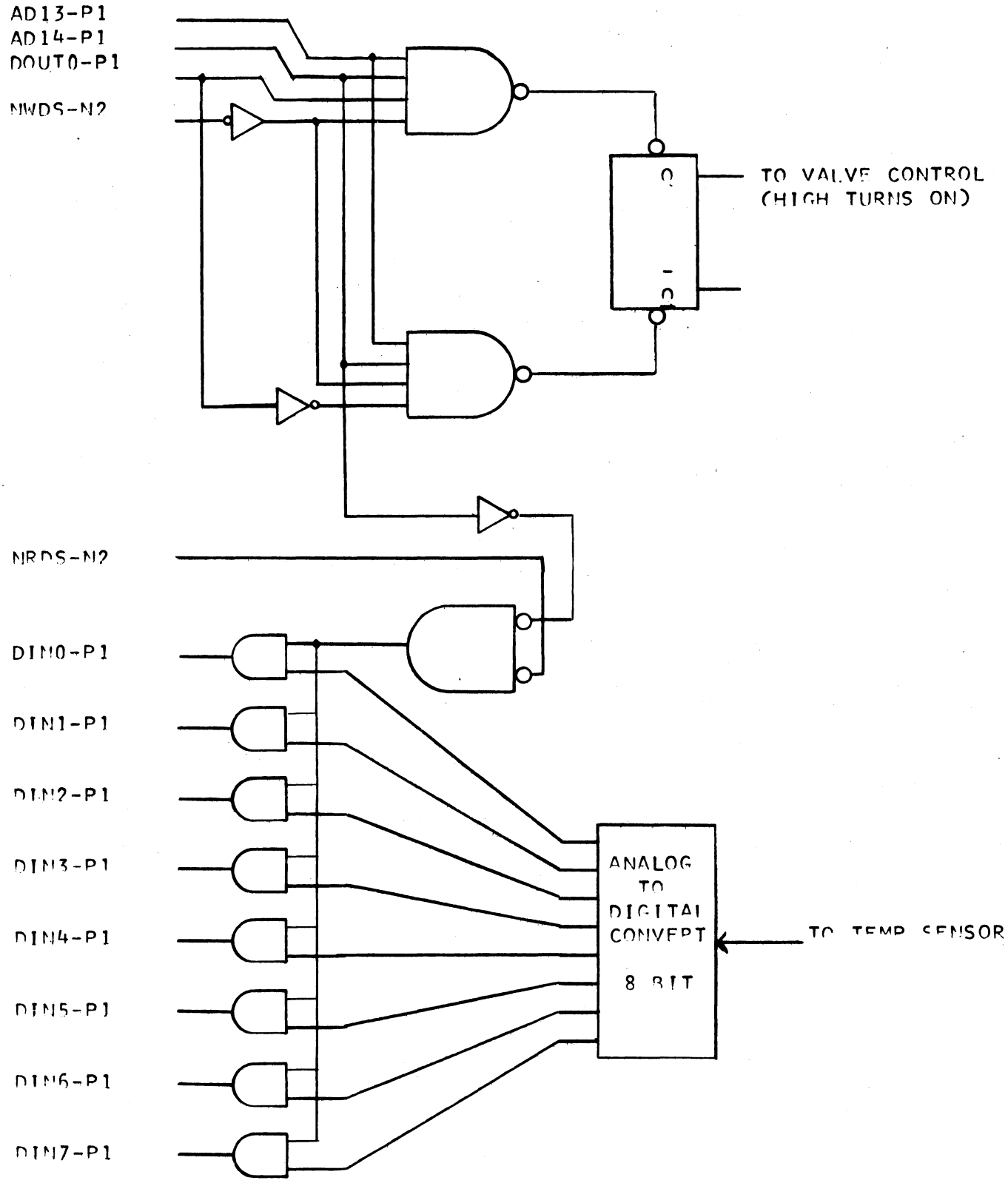
CONSIDER THE FOLLOWING PROBLEM. I WANT TO CONTROL THE TEMPERATURE OF A PROCESS, IF THE TEMPERATURE REACHES A CERTAIN VALUE, I WANT TO OPEN A VALVE, TO INCREASE FLOW AND REDUCE THE TEMPERATURE. THE TEMPERATURE SENSOR A/D INPUT IS ADDRESSED AT MEMORY LOCATION 16384. THE TEMPERATURE IS SCALED IN A LINEAR FASHION FROM 0 TO 255 F OVER THE RANGE OF AN 8 BIT A/D. IF FOR EXAMPLE I READ 128 FROM THE A/D I HAVE A TEMPERATURE OF 128 F. I WANT TO DETECT A TEMPERATURE OF 128 F AND TURN ON A VALVE ADDRESSED AT MEMORY LOCATION 24576. HERE IS THE PROGRAM FOR THE CONTROL FUNCTION.

```
10 LET X=@16384                 READ MEMORY LOCATION 16384 INTO X
20 IF X>=128 GOTO 30            IF TEMP IS = OR GREATER THAN 128
                                GOTO STATEMENT 30
25 GOTO 10                      SINCE TEMP IS LOWER THAN 128 GO
                                BACK UP AND READ IT AGAIN.
30 LET @24576=255               WRITE ALL 1'S TO LOCATION 24576,
                                ANY ONE OF WHICH CAN BE USED TO
                                TURN ON THE VALVE.
40 PRINT "THE VALVE IS OPEN"
50 END
```

HOW ABOUT CLOSING THE VALVE AFTER THE PROCESS COOLS OFF A BIT? LETS CHANGE THE PROGRAM A LITTLE

```
10 LET X=@16384
20 IF X<100 GOTO 35             IF THE TEMP OF THE PROCESS
                                IS BELOW 100 F BE SURE THE
                                VALVE IS TURNED OFF
30 IF X>=128 GOTO 40            IF THE TEMPERATURE OF THE PROCESS
                                IS ABOVE OR EQUAL TO 128 F,
                                TURN ON THE VALVE.
32 GOTO 10                      GO CHECK THE TEMP AGAIN
35 LET@24576=0 : GOTO 10        TURN OFF VALVE, GO CHECK TEMP AGAIN
40 LET @24576=255 : GOTO 10     TURN ON VALVE, GO CHECK TEMP AGAIN
```

I AM READING LOCATION 16384, IF TEMP IS LESS THAN 100 I TURN THE VALVE OFF, IF THE TEMP IS EQUAL TO OR ABOVE 128, I WILL TURN THE VALVE ON. IN THE REGION 100 TO 127, THE VALVE WILL REMAIN IN THE PRESENT CONDITION.



HARDWARE TO SUPPORT APPLICATION PROGRAM

3.5.4 A LOOK AT THE EASE OF I/O PROGRAMMING

LETS CRAWL DOWN INSIDE OF THE NIBBLER AND LOOK AT SOME OF THE I/O SIGNALS GENERATED WITH SOME RATHER SIMPLE PROGRAMS. THE WAVEFORMS SHOWN HERE WERE OBTAINED AS FOLLOWS.

THE SCOPE WAS TRIGGERED OFF OF ADDRESS LINE AD14-P1. WHY USE THIS LINE? WE WANT A SWEEP TO BE TRIGGERED AT A REPETITIVE RATE BUT NOT RECURRING MORE OFTEN THAN WHAT WE WANT TO LOOK AT. AD14-P1 IS THE ADDRESS LINE FOR THE 16K MEMORY BOUNDARY. OUR EXISTING ROM & RAM ARE BELOW THIS ADDRESS, THEREFORE THIS ADDRESS LINE SHOULD NEVER BE TRUE EXCEPT IF WE ADDRESS IT THROUGH OUR PROGRAM. BY PLACING A MEMORY ACCESS INSTRUCTION IN OUR PROGRAM WE CAN USE AD14-P1 AS A MILESTONE MARKER.

THE TRACES FOR OBSERVATION WOULD BE:

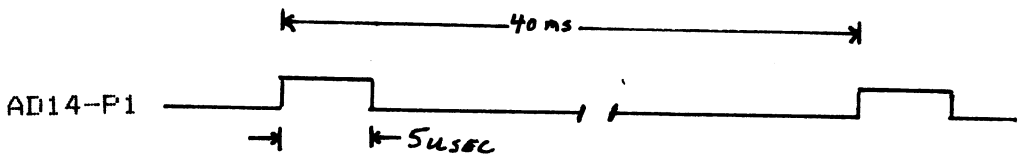
A. NADS--ADDRESS STROBE
B. NWDS--WRITE STROBE
C. NRDW--READ STROBE
D. DOUT(0-7)--DATA OUT LINES

THE FOLLOWING PROGRAM WAS WRITTEN:

```
10 LET X=@16384 : GOTO 10
```

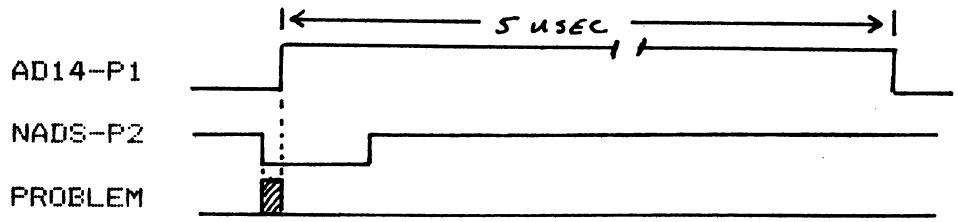
THIS PROGRAM READS THE CONTENTS OF MEMORY LOCATION 16384 INTO THE VARIABLE X, THEN BRANCHES BACK TO REPEAT THE STATEMENT.

THE FOLLOWING WAVEFORM RESULTS:

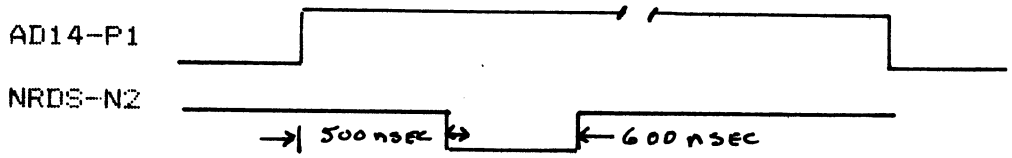


THE FOLLOWING CONCLUSION COULD BE REACHED:

THE EXECUTION OF THE TWO STATEMENTS ON LINE 10 REQUIRE 40 MSEC. THE WIDTH OF THE AD14-P1 PULSE IS 5 USEC, WHAT IS THE REASON FOR THIS? RECALL THAT ADDRESS LINES AD(12, 13, 14, & 15) ARE LOCATED IN THE 74LS174 FROM THE LEAST SIGNIFICANT DATA BUS BITS DB(0-3). THEREFORE THE AD14 LINE WILL STAY TRUE UNTIL A NEW ADDRESS VALUE IS PLACED ON THE ADDRESS LINES. LET'S LOOK AT THE ADDRESS STROBE NADS-N2 IN RELATION TO AD14-P1.



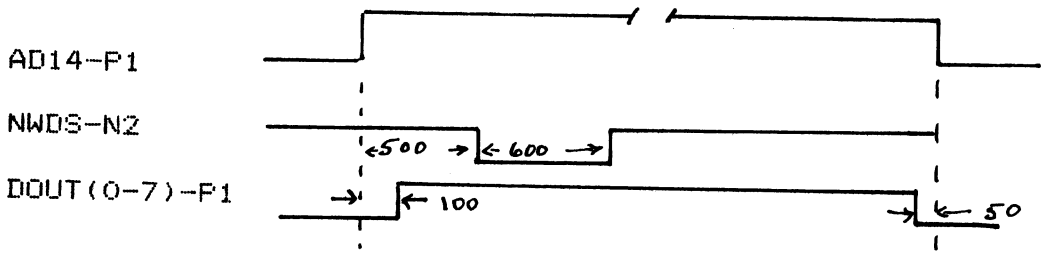
NOTICE THE LITTLE SLIVER THAT IS LABELED PROBLEM. THIS TIME IS ABOUT 50 NSEC. THE PROBLEM IS THE TIME THAT THE ADDRESS STROBE APPEARS BEFORE THE ADDRESS IS CORRECT. RECALL THE LATCHING OF THE AS(12-14) BITS IN THE 74LS174. THE STROBE USED TO CLOCK THE DATA BUS BITS(0-3) INTO THE 74LS174 IS NADS-P2. THEREFORE, THERE IS A FINITE DELAY CAUSED BY THE INVERTER PROPAGATION DELAY AND THE 74LS174 PROPAGATION DELAY BEFORE THE ADDRESS IS FIRM IN THE 74LS174 OUTPUTS. THE REASON FOR THIS DISCUSSION IS TO SHOW WHY YOU MUST USE THE TRAILING EDGE OF NADS-N2 FOR CLOCKING THE ADDRESS INTO A FLIP FLOP. IF YOU USE A LATCH LIKE A 7475 THERE WOULD BE NO PROBLEM AS THE DATA OUT WILL FOLLOW THE DATA IN UNTIL THE CLOCK PULSE GOES FALSE. LET'S LOOK AT THE READ STROBE NRDS-P2 IN RELATION TO AD14-P1.



THE PROGRAM COULD BE CHANGED TO THE FOLLOWING:

```
10 LET@16384=255 : GOTO 10
```

THIS PROGRAM WRITES ALL ONES TO LOCATION 16384. LETS LOOK AT THE WRITE STROBE NWDS-P2 IN RELATION TO AD14-P1 AND DOUT (0-7)-P1.



THE DATA ARE FIRM DURING THE STROBE TIME AND COULD BE LATCHED INTO A DATA REGISTER. LET'S GO DOWN ONE MORE LEVEL OF PROGRAMMING TO THE MACHINE LANGUAGE OF THE SC/MP.

```
LETS WRITE THE FOLLOWING PROGRAM:
10 R=5000                          BUFFER AREA FOR 2 CHAR. STRING.
20 INPUT U                         STARTING LOC. FOR INPUT
40 INPUT $R                        INPUT TWO HEX CHAR. IN A STRING.
50 LET X=@(R) : GOSUB 1000         CONVERT FIRST DIGIT
55 A=X
60 LET X=@(R+1): GOSUB 1000        CONVERT SECOND DIGIT
65 B=X
70 C=(16*A)+B                      CONVERT 2 DIGIT NBR INTO DECIMAL
80 LET @(U)=C                      WRITE DECIMAL BYTE
90 U=U+1
100 GOTO 40                        GO DO ANOTHER ONE
1000 IF X=65 X=58                  IS IT "A", SET = 10
1010 IF X=66 X=59                  IS IT "B", SET = 11
1020 IF X=67 X=60                  IS IT "C", SET =12
1030 IF X=68 X=61                  IS IT "D", SET = 13
1040 IF X=69 X=62                  IS IT "E", SET = 14
1050 IF X=70 X=63                  IS IT "F", SET = 15
1060 X=X-48                        CONVERT 0-9
1070 RETURN
```

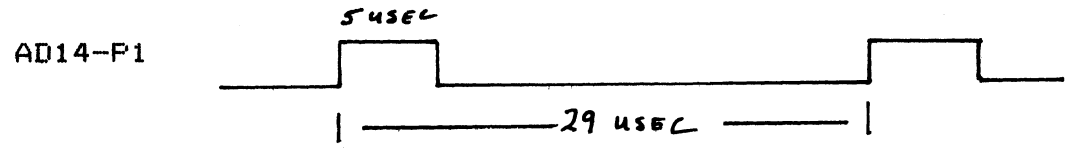
THIS PROGRAM ALLOWS US TO SPECIFY A LOCATION IN MEMORY AND TO INPUT HEXADECIMAL BYTES INTO THE LOCATION SPECIFIED AND INTO FOLLOWING SEQUENTIAL MEMORY LOCATIONS.

EXECUTING THE PROGRAM: INPUT LOCATION 6000 AS START OF DATA INPUT. THEN INPUT THE FOLLOWING STRINGS C4, 40, 35, C4, 00, 31, C9, 00, 90, FC.

THIS CHARACTER STRING IS A MACHINE LANGUAGE PROGRAM AS DESCRIBED BELOW

ADDRESS	MACHINE CODE	
6000	C440	LOAD A REG WITH 40 (HEX)
6002	35	EXCHANGE A WITH P1 REG HIGH BYTE
6003	C400	LOAD A REG WITH 00
6005	31	EXCHANGE A WITH PA REG LOW BYTE
6006	C900	STORE A INTO LOC. INDEXED BY P1
6008	90FC	JUMP BACK TO LOC 6006

THE P REGISTER WILL CONTAIN LOCATION 4000 HEX OR 16384 DECIMAL. WHEN THIS PROGRAM IS EXECUTED THE A REGISTER WILL BE STORED REPEATEDLY INTO MEMORY LOCATION 16384. THIS PROGRAM IS SIMILAR TO THE NIBL PROGRAM BUT IN MACHINE LANGUAGE. EXECUTE THE PROGRAM BY USING THE NIBL LINK STATEMENT. "LINK 6000." AGAIN LOOK AT AD14-P1.

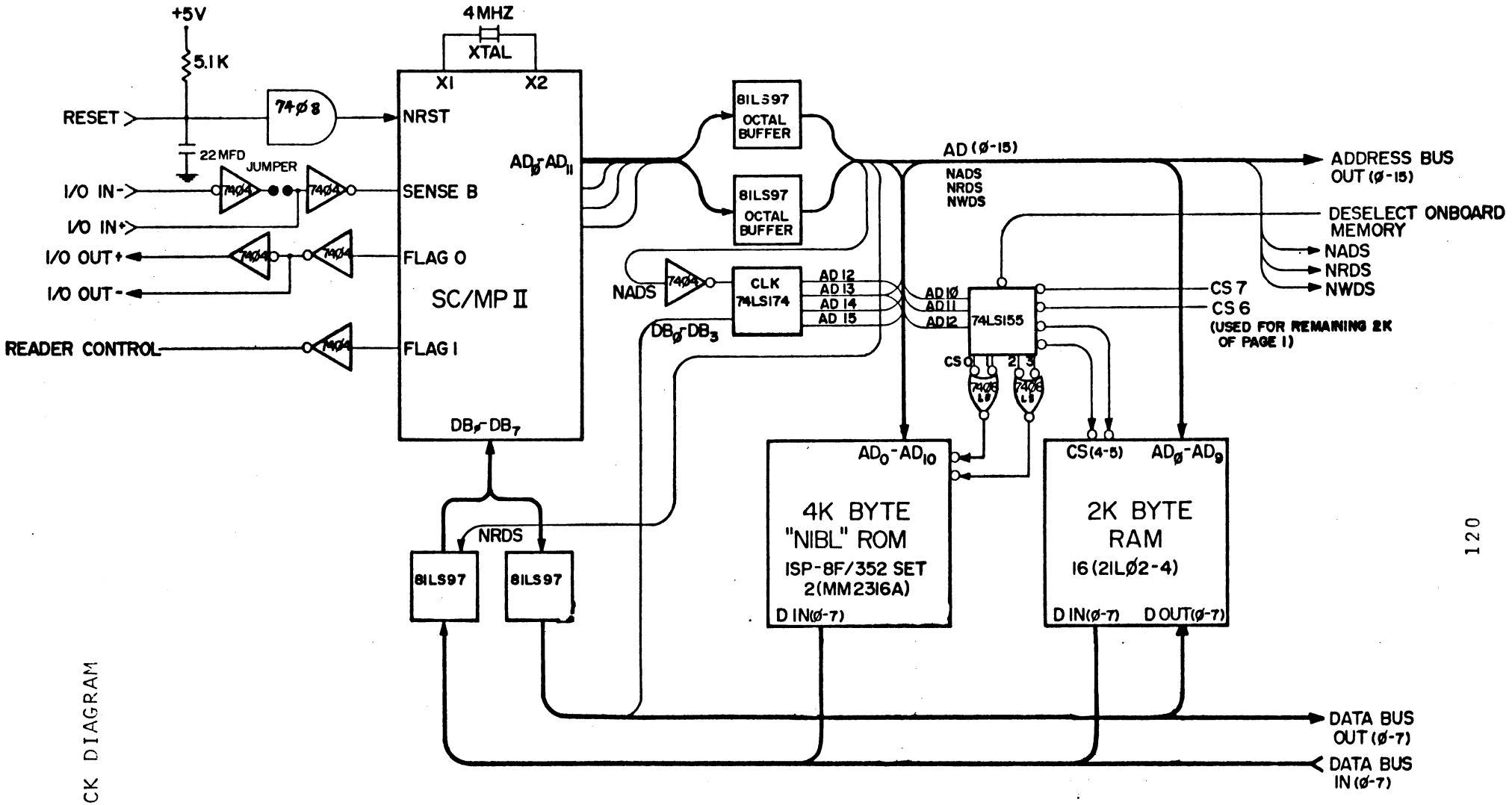


THE STORE ACCUMULATOR INSTRUCTION REQUIRES 18 MICROCYCLES FOR EXECUTION TIME. A MICROCYCLE IS EQUAL TO 1 USEC IN A SC/MP II RUNNING AT 4 MHZ. A JUMP INSTRUCTION REQUIRES 11 MICROCYCLES, THEREFORE THE 29 USEC IS THE TOTAL EXECUTION TIME OF THE LOOP WE PROGRAMMED.

```
ST     18 USEC.
JMP    11 USEC.
       -------
       29 USEC.
```

THE SPEED ADVANTAGE OF MACHINE LANGUAGE PROGRAMS OVER NIBL PROGRAMS CAN EASILY BE SEEN. YOU MAY WRITE MACHINE LANGUAGE SUBROUTINES FOR APPLICATIONS THAT REQUIRE SPEED, USING THE LINK STATEMENT TO CALL THE MACHINE LANGUAGE PROGRAM. USE THE CONVENTION SHOWN IN THE NIBL MANUAL TO RETURN TO YOUR NIBL PROGRAM.

3.6 LOGIC DIAGRAMS

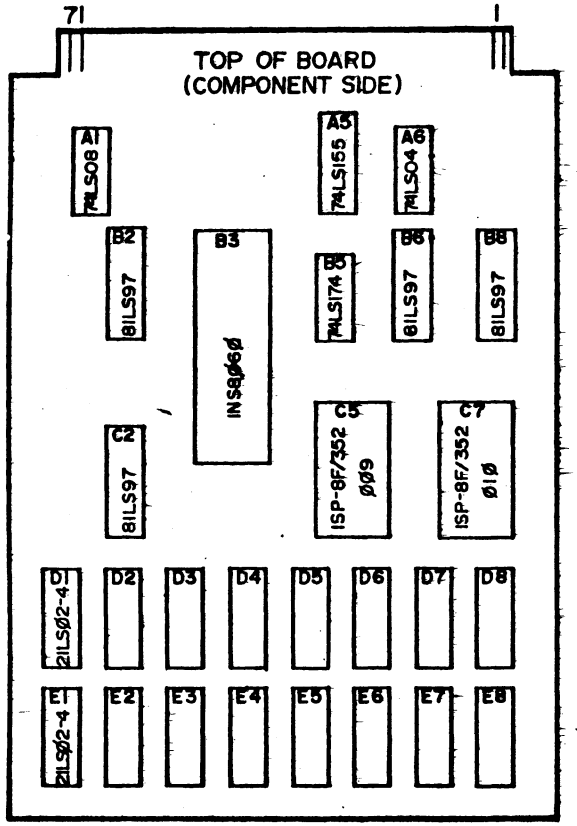


3.6.1 BLOCK DIAGRAM

120

BLOCK DIAGRAM
NIBBLER
SHEET 1 OF 5 REV 1

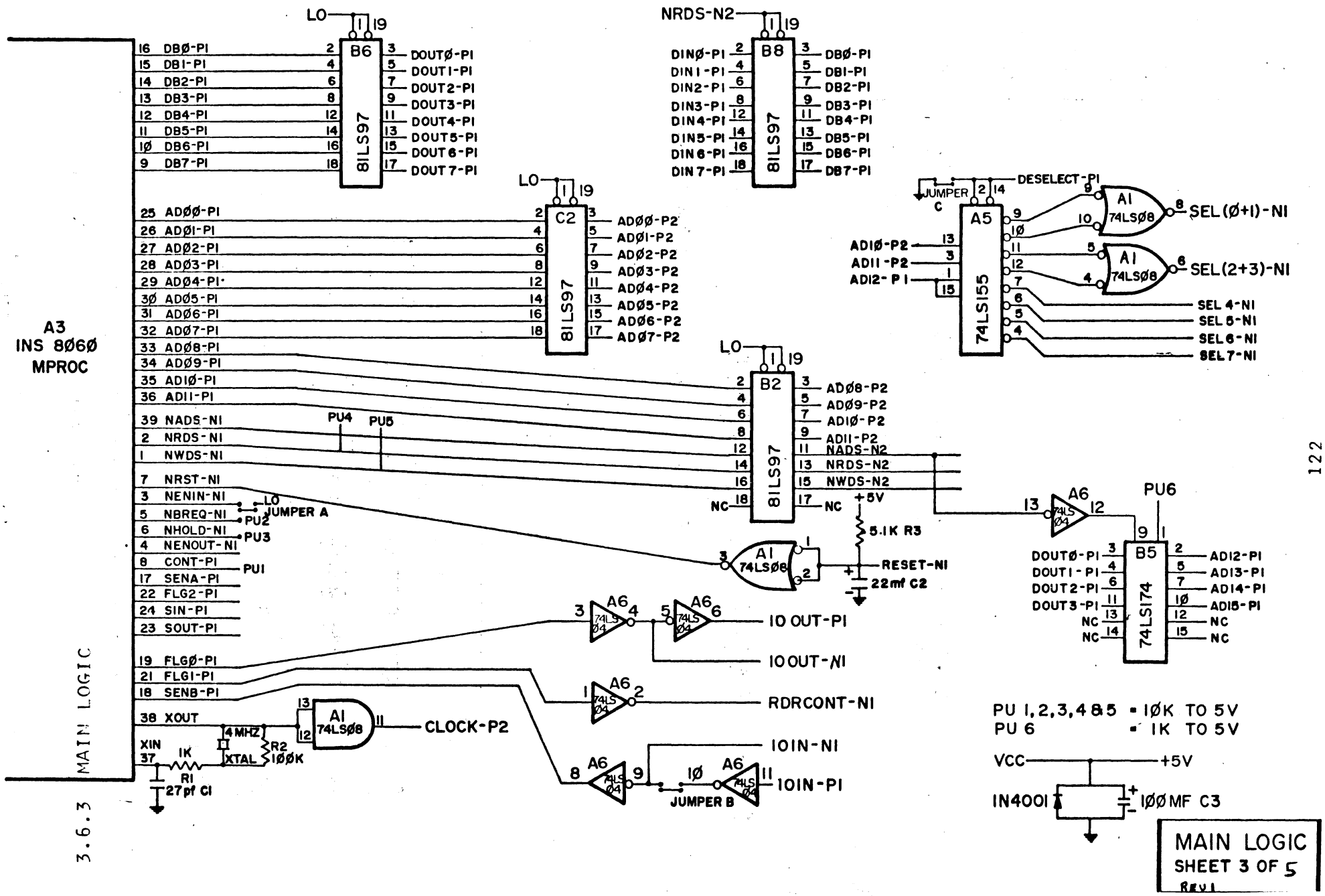
CHIP/BOARD MAP

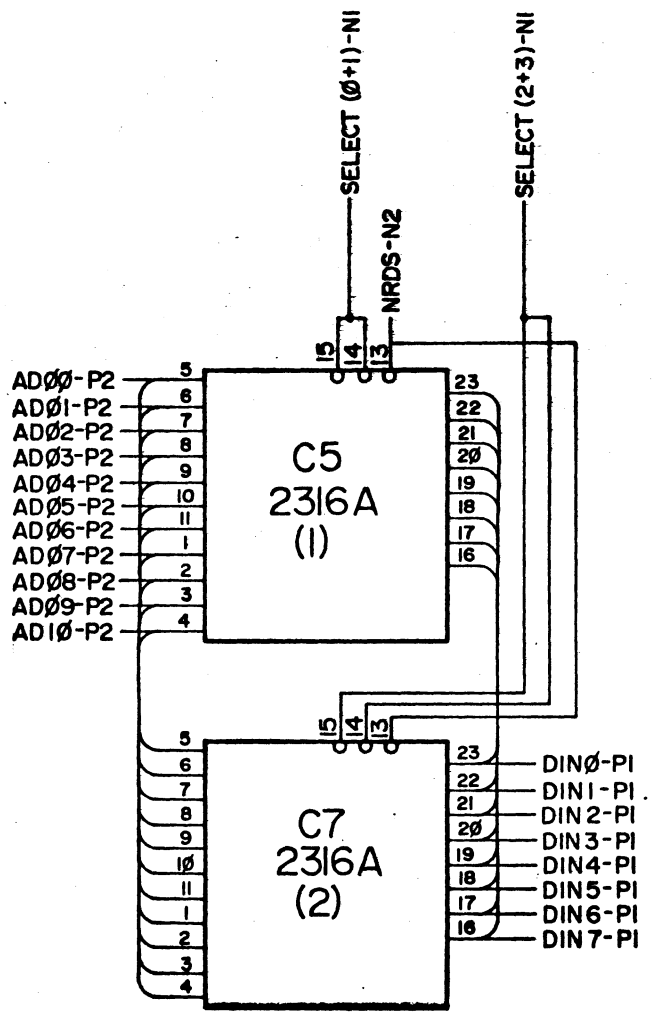


EDGE CONNECTOR MAP

1	+5	37	AD12-PI
2	+5	38	AD13-PI
3	DIN7-PI	39	NRDS-N2
4	IO OUT-PI	40	SIN-PI
5	DIN6-PI	41	SPARE
6	IOIN-PI	42	SOUT-PI
7	DIN5-PI	43	SPARE
8	IOOUT-NI	44	FLG2-PI
9	DIN4-PI	45	SPARE
10	IOIN-NI	46	SENA-PI
11	DIN3-PI	47	SPARE
12	RDRCONT-NI	48	NENOUT-NI
13	DIN2-PI	49	RESET-NI
14	DOUT7-PI	50	NENIN-NI
15	DIN1-PI	51	NBREQ-NI
16	DOUT6-PI	52	NHOLD-NI
17	DINØ-PI	53	CONT-PI
18	DOUT5-PI	54	NADS-N2
19	SPARE	55	SPARE
20	DOUT4-PI	56	NWDS-N2
21	SPARE	57	SPARE
22	DOUT3-PI	58	CLOCK-P2
23	SPARE	59	AD11-P2
24	DOUT2-PI	60	AD1Ø-P2
25	SPARE	61	ADØ9-P2
26	DOUT1-PI	62	ADØ8-P2
27	DESELECT-PI	63	ADØ7-P2
28	DOUTØ-PI	64	ADØ6-P2
29	SEL6-NI	65	ADØ5-P2
30	SEL7-NI	66	ADØ4-P2
31	-12V	67	ADØ3-P2
32	-12V	68	ADØ2-P2
33	+12V	69	ADØ1-P2
34	+12V	70	ADØØ-P2
35	AD14-PI	71	GND
36	AD15-PI	72	GND

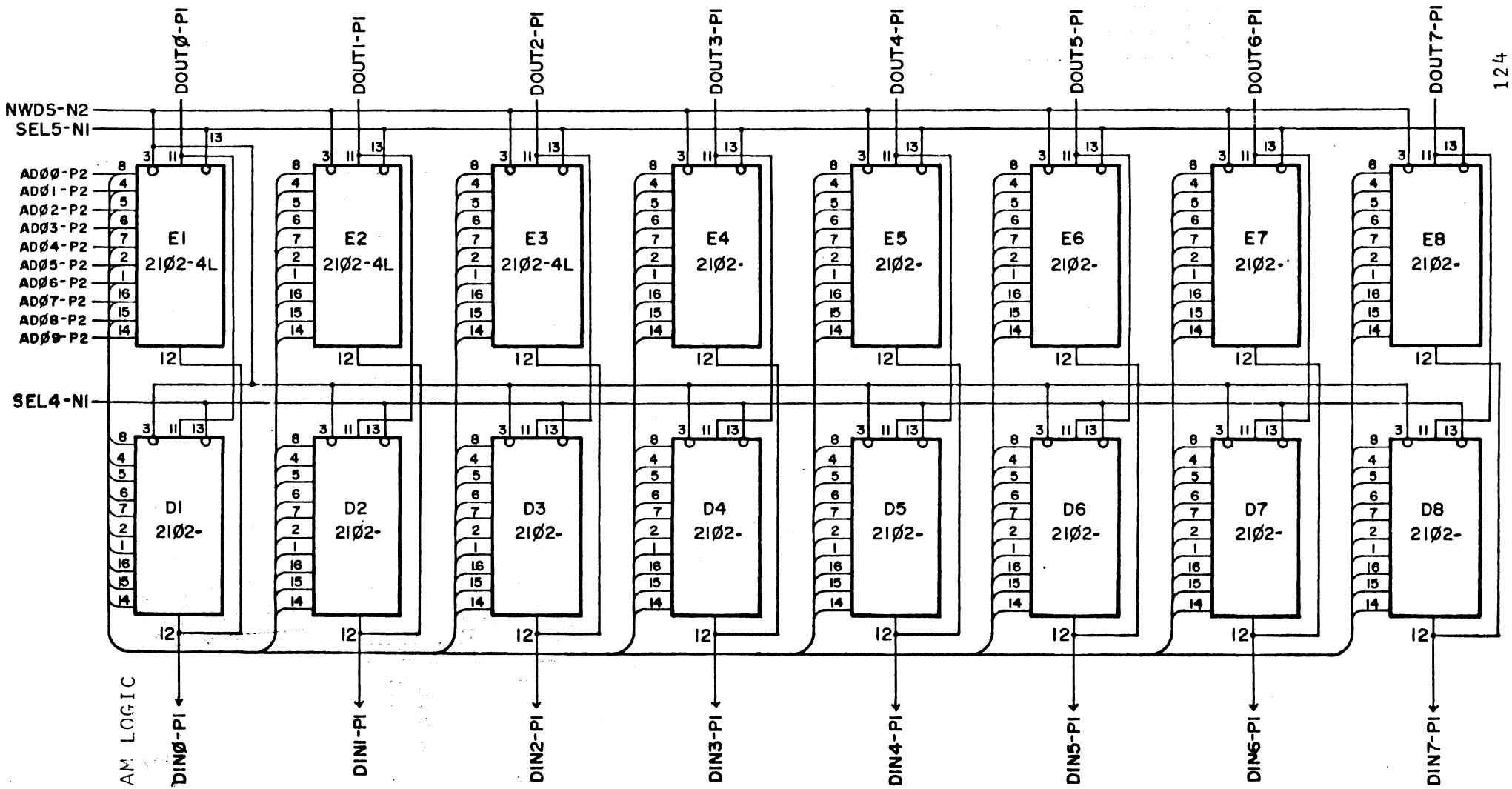
BOARD AND EDGE CONNECTOR MAP
SHEET 2 OF 5 REV 1


A3
INS 8Ø6Ø
MPROC
MAIN LOGIC
3.6.3
B6
81LS97
B8
81LS97
C2
81LS97
B2
81LS97
A5
74LS155
B5
74LS174
A1
74LSØ8
A6
74LSØ4
JUMPER A
JUMPER B
JUMPER C
4 MHZ
XTAL
R1 1K
R2 1ØØK
C1 27pf
C2 22mf
C3 1ØØMF
R3 5.1K
IN4001
VCC
+5V
DESELECT-PI
SEL(Ø+1)-NI
SEL(2+3)-NI
SEL 4-NI
SEL 5-NI
SEL 6-NI
SEL 7-NI
RESET-NI
CLOCK-P2
RDRCONT-NI
IO OUT-PI
IOOUT-NI
IOIN-NI
IOIN-PI
PU1
PU2
PU3
PU4
PU5
PU6
PU 1,2,3,4&5 = 1ØK TO 5V
PU 6 = 1K TO 5V
MAIN LOGIC
SHEET 3 OF 5
REV1
122



THESE ARE PRE-PROGRAMMED TO CONTAIN NIBL. ROM 1 IS LOWER 2K, ROM 2 IS UPPER 2K.

4K ROM (NIBL)
SHEET 4 OF 5 REV 1



3.6.5 RAM LOGIC

2K RAM
SHEET 5 OF 5
REV 1

3.6.6 WIRE LIST A CHIPS

A1 74LS08

1. RESET-N1, A1-2, R3, C2, EC-49
2. RESET-N1, A1-1, R3, C2, EC-49
3. NRST-N1, A3-7
4. SEL3-N1, A5-12
5. SEL2-N1, A5-11
6. SEL(2+3)-N1, C7-14, C7-15
7. GND
8. SEL(0+1)-N1, C5-14, C5-15
9. SEL 0-N1, A5-9
10. SEL 1-N1, A5-10
11. CLOCK-P2, EC-58
12 XOUT, A1-13, R2, XTAL, A3-38
13. XOUT, A1-12, R2, XTAL, A3-38
14. VCC

A5 74LS155

1. AD12-P1, A5-15, B5-2, EC-37
2. DESELECT-P1, A5-14, JUMPER C, EC-27
3. AD11-P2, B2-9, EC-59
4. SEL7-N1, EC-30
5. SEL6-N1, EC-29
6. SEL5-N1, (E1-E8)-13
7. SEL4-N1, (D1-D8)-13
8. GND
9. SEL0-N1, A1-9
10. SEL1-N1, A1-10
11. SEL2-N1, A1-5
12. SEL3-N1, A1-4
13. AD10-P2, B2-7, EC-60, C5-4, C7-4
14. DESELECT-P1, A5-2, JUMPER C, EC-27
15. AD12-P1, A5-1, B5-2, EC-37
16. VCC

A6 74LS04

1. FLG1-P1, A3-21
2. RDRCONT-N1, EC-12
3. FLG0-P1, A3-19
4. IO OUT-N1, A6-5, EC-8
5. IO OUT-N1, A6-4, EC-8
6. IO OUT-P1, A6-6, EC-4
7. GND
8. SENB-P1, A3-13
9. IO1N-N1, JUMPER B, TO A6-10, EC-6
10. IOIN-N1, JUMPERB, TO A6-9, EC-6
11. IOIN-P1, EC-10
12. NADS-P2, B5-9
13. NADS-N2, B2-11, EC-54
14. VCC

A3 INS8060

1. NWDS-N1, PU5, B2-16
2. NRDS-N1, PU4, B2-14
3. NENIN-N1, JUMPER A, JUMPED LOW NORMALLY, EC-50
4. NENOUT-N1, EC-48
5. NBREQ-N1, PU2, EC-51
6. NHOLD-N1, PU3, EC-52
7. NRST-N1, A1-3
8. CONT-P1, PU1, EC-53
9. DB7-P1, B6-18, B8-17
10. DB6-P1, B6-16, B8-15
11. DB5-P1, B6-14, B8-13
12. DB4-P1, B6-12, B8-11
13. DB3-P1, B6-8, B8-9
14. DB2-P1, B6-6, B8-7
15. DB1-P1, B6-4, B8-5
16. DB0-P1, B6-2, B8-3
17. SENA-P1, EC-46
18. SENB-P1, A6-8
19. FLG0-P1, A6-3
20. GND
21. FLG1-P1, A6-1
22. FLG2-P1, EC-44
23. SOUT-P1, EC-42
24. SIN-P1, EC-40
25. AD00-P1, C2-2
26. AD01-P1, C2-4
27. AD02-P1, C2-6
28. AD03-P1, C2-8
29. AD04-P1, C2-12
30. AD05-P1, C2-14
31. AD06-P1, C2-16
32. AD07-P1, C2-18
33. AD08-P1, B2-2
34. AD09-P1, B2-4
35. AD10-P1, B2-6
36. AD11-P1, B2-8
37. XIN, C1, R1
38. XOUT, XTAL, R2, A1-12, A1-13
39. NADS-N1, B2-12
40. VCC

WIRE LIST B CHIPS

B2 81LS97

1. GND
2. AD08-P1, A3-33
3. AD08-P2, C5-2, C7-2, (D1-D7)-15, (E1-E7)-15, EC-62
4. AD09-P1, A3-34
5. AD09-P2, C5-3, C7-3, (D1-D7)-14, (E1-E7)-14, EC-61
6. AD10-P1, A3-35
7. AD10-P2, C5-4, C7-4, EC-60, A5-13
8. AD11-P1, A3-36
9. AD11-P2, A5-3, EC-59
10. GND
11. NADS-N2, A6-13, EC-54
12. NADS-N1, A3-39
13. NRDS-N2, C5-13, C7-13, EC-39
14. NRDS-N1, PU4, A3-2
15. NWDS-N2, (E1-E7)-3, (D1-D7)-3, EC-56
16. NWDS-N1, PU5, A3-1
17. NC
18. NC
19. GND
20. VCC

B5 74LS174

1. PU 6
2. AD12-P1, EC-37
3. DOUT0-P1, B6-3, D1-11, E1-11, EC-28
4. DOUT1-P1, B6-5, D2-11, E2-11, EC-26
5. AD13-P1, EC-38
6. DOUT2-P1, B6-7, D3-11, E3-11, EC-24
7. AD14-P1, EC-35
8. GND
9. NADS-P2, A6-13
10. AD15-P1, EC-36
11. DOUT3-P1, B6-9, D4-11, E4-11, EC-22
12. NC
13. NC
14. NC
15. NC
16. VCC

B6 81LS97

1. GND
2. DB0-P1, A3-16, B8-3
3. DOUT0-P1, B5-3, E1-11, D1-11, EC-28
4. DB1-P1, A3-15, B8-5
5. DOUT1-P1, B5-4, E2-11, D2-11, EC-26
6. DB2-P1, A3-14, B8-7
7. DOUT2-P1, B5-6, E3-11, D3-11, EC-24
8. DB3-P1, A3-13, B8-9
9. DOUT3-P1, B5-11, E4-11, D4-11, EC-22
10. GND
11. DOUT4-P1, E5-11, D5-11, EC-20
12. DB4-P1, A3-12, B8-11
13. DOUT5-P1, E5-11, D5-11, EC-18
14. DB5-P1, A3-11, B8-13
15. DOUT6-P1, E7-11, D7-11, EC-16
16. DB6-P1, A3-10, B8-15
17. DOUT7-P1, E8-11, D8-11, EC-14
18. DB7-P1, A3-9, B8-17
19. GND
20. VCC

B8 71LS97

1. NRDS-N2
2. DIN0-P1, C5-23, C7-34, D1-12, E1-12, EC-17
3. DB0-P1, A3-16, B6-2
4. DIN1-P1, C5-22, C7-22, D2-12, E2-12, EC-15
5. DB1-P1, A3-15, B6-4
6. DIN2-PA, C5-21, C7-21, D2-12, E3-12, EC-13
7. DB2-P1, A3-14, B6-6
8. DIN3-P1, C5-20, C7-20, D4-12, E4-12, EC-11
9. DB3-P1, A3-13, B6-8
10. GND
11. DB4-P1, A3-12, B6-12
12. DIN4-P1, C5-19, C7-19, D5-12, E5-12, EC-9
13. DB5-P1, A3-11, B6-14
14. DIN5-P1, C5-18, C7-18, D6-12, E6-12, EC-7
15. DB6-P1, A3-10, B6-16
16. DIN6-P1, C5-17, C7-17, D7-12, E7-12, EC-5
17. DB7-P1, A3-9, B6-18
18. DIN7-P1, C5-16, C7-16, D8-12, E8-12, EC-3
19. NRDS-N2
20. VCC

3.6.8 WIRE LIST C CHIPS

C2 81LS97

1. GND
2. AD00-P1, A3-25
3. AD00-P1, C5-5, C7-5, (D1-D7)-8, (E1-E7)-8, EC-70
4. AD01-P1, A3-26
5. AD01-P2, C5-6, C7-6, (D1-D7)-4, (E1-E7)-4, EC-69
6. AD02-P1, A3-27
7. AD02-P2, C5-7, C7-7, (D1-D7)-5, (E1-E7)-5, EC-68
8. AD03-P1, A3-28
9. AD03-P2, C5-8, C7-8, (D1-D7)-6, (E1-E7)-6, EC-67
10. GND
11. AD04-P2, C5-9, C7-9, (D1-D7)-7, (E1-E7)-7, EC-66
12. AD04-P1, A3-29
13. AD05-P2, C5-10, C7-10, (D1-D7)-2, (E1-E7)-2, EC-65
14. AD05-1, A3-30
15. AD06-P2, C5-11, C7-11, (D1-D7)-1, (E1-E7)-1, EC-64
16. AD06-P1, A3-31
17. AD07-P2, C5-1, C7-1, (D1-D7)-16, (E1-E7)-16, EC-63
18. AD07-P1, A3-32
19. GND
20. VCC

C5 2316A (ROM1)

1. AD07-P2 NET
2. AD08-P2 NET
3. AD09-P2 NET
4. AD10-P2, B2-7, EC-60
5. AD00-P2 NET
6. AD01-P2 NET
7. AD02-P2 NET
8. AD03-P2 NET
9. AD04-P2 NET
10. AD05-P2 NET
11. AD06-P2 NET
12. GND
13. NRDS-N2, B2-12, EC-39
14. SEL (0+1)-N1, A1-8
15. SEL (0+1)-N1, A1-8
16. DIN7-P1 NET
17. DIN6-P1 NET
18. DIN5-P1 NET
20. DIN3-P1 NET
21. DIN2-P1 NET
22. DIN1-P1 NET
23. DIN0-P1 NET
24. VCC

C7 2316A (ROM2)

1. AD07-P2 NET
2. AD08-P2 NET
3. AD09-P2 NET
4. AD10-P2, B2-7, EC-60
5. AD00-P2 NET
6. AD01-P2 NET
7. AD02-P2 NET
8. AD03-P2 NET
9. AD04-P2 NET
10. AD05-P2 NET
11. AD06-P2 NET
12. GND
13. NRDS-N2, B2-13, EC-39
14. SEL (2+3)-N1, A1-6
15. SEL (2+3)-N1, A1-6
16. DIN7-P2 NET
17. DIN6-P2 NET
18. DIN5-P2 NET
19. DIN4-P2 NET
20. DIN3-P2 NET
21. DIN2-P2 NET
22. DIN1-P2 NET
23. DIN0-P2 NET
24. VCC

D(1-7) 2102

1. AD06-P2 NET
2. AD05-P2, NET
3. NWDS-N2, B2-15, EC-56
4. AD01-P2 NET
5. AD02-P2 NET
6. AD03-P2 NET
7. AD04-P2 NET
8. AD00-P2 NET
9. GND
10. VCC
11. (DOUT0-P1)-(DOUT7-P1) NET
12. (DIN0-P1)-(DIN7-P1) NET
13. SEL4-N1, A5-7
14. AD09-P2 NET
15. AD08-P2 NET
16. AD07-P2 NET

E(1-7) 2102

1. AD06-P2 NET
2. AD05-P2 NET
3. NWDS-N2, B2-15, EC-56
4. AD01-P2 NET
5. AD02-P2 NET
6. AD03-P2 NET
7. AD04-P2 NET
8. AD05-P2 NET
9. GND
10. VCC
11. (DOUT0-P1)-(DOUT7-P1) NET
12. (DIN0-P1)-(DIN7-P1) NET
13. SEL5-N1, A5-6
14. AD09-P2 NET
15. AD08-P2 NET
16. AD07-P2 NET

3.6.9 WIRE LIST LARGE SIGNAL NETS

ADDRESS LINE NET

AD00-P2 NET C5-5, C7-5, C2-3, (D1-D7)-8, (E1-E7)-8, EC-70
AD01-P2 NET C5-6, C7-6, C2-5, (D1-D7)-4, (E1-E7)-4, EC-69
AD02-P2 NET C5-7, C7-7, C2-7, (D1-D7)-5, (E1-E7)-5, EC-68
AD03-P2 NET C5-8, C7-8, C2-9, (D1-D7)-6, (E1-E7)-6, EC-67
AD04-P2 NET C5-9, C7-9, C2-11, (D1-D7)-7, (E1-E7)-7, EC-66
AD05-P2 NET C5-10, C7-10, C2-13, (D1-D7)-2, (E1-E7)-2, EC-65
AD06-P2 NET C5-11, C7-11, C2-15, (D1-D7)-1, (E1-E7)-1, EC-64
AD07-P2 NET C5-1, C7-1, C2-17, (D1-D7)-16, (E1-E7)-16, EC-63
AD08-P2 NET C5-2, C7-2, B2-3, (D1-D7)-15, (E1-E7)-15, EC-62
AD09-P2 NET C5-3, C7-3, B2-5, (D1-D7)-14, (E1-E7)-14, EC-61

DATA OUT LINE NET

DOUT0-P1 NET B6-3, B5-3, E1-11, D1-11, EC-28
DOUT1-P1 NET B6-5, B5-4, E2-11, D2-11, EC-26
DOUT2-P1 NET B6-7, B5-6, E3-11, D3-11, EC-24
DOUT3-P1 NET B6-9, B5-11, E4-11, D4-11, EC-22
DOUT4-P1 NET B6-11, E5-11, D5-11, EC-20
DOUT5-P1 NET B6-13, E6-11, D6-11, EC-18
DOUT6-P1 NET B6-15, E7-11, D7-11, EC-16
DOUT7-P1 NET B6-17, E8-11, D8-11, EC-14

DATA IN LINE NET

DIN0-P1 NET B8-2, C5-23, C7-23 D1-12, E1-12, EC-17
DIN1-P1 NET B8-4, C5-22, C7-22, D2-12, E2-12, EC-15
DIN2-P1 NET B8-6, C5-21, C7-21, D3-12, E3-12, EC-13
DIN3-P1 NET B8-8, C5-20, C7-20, D4-12, E4-12, EC-11
DIN4-P1 NET B8-12, C5-19, C7-19, D5-12, E5-12, EC-9
DIN5-P1 NET B8-14, C5-18, C7-18, D6-12, E6-12, EC-7
DIN6-P1 NET B8-16, C5-17, C7-17, D7-12, E7-12, EC-5
DIN7-P1 NET B8-18, C5-16, C7-16, D8-12, E8-12, EC-3

3.7 SC/MP AND NIBL INTEGRATED CIRCUIT TECHNICAL DESCRIPTIONS

3.7.1 ISP-8A/600 (SC/MP II)
3.7.2 MM2316A 16K ROM (NIBL ROM)

National Semiconductor

JUNE 1977

MM2316A 16,384-bit read only memory

general description

The MM2316A is a static MOS 16,384-bit read-only memory organized in a 2048-word-by-8-bit format. It is fabricated using N-channel enhancement and depletion-mode silicon-gate technology which provides complete DTL/TTL compatibility and single power-supply operation.

Three programmable chip selects controlling the TRI-STATE® outputs allow for memory expansion.

Programming of the memory array and chip-select active levels is accomplished by changing one mask during fabrication.

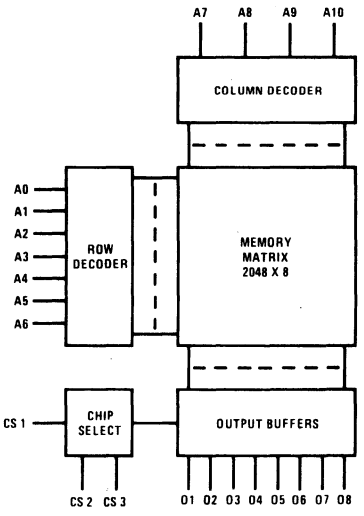
features

- Fully decoded
- Single 5V power supply
- Inputs and outputs TTL compatible
- Static operation
- TRI-STATE outputs for bus interface
- Programmable chip selects
- 2048 word by 8-bit organization
- Maximum access time—450 ns

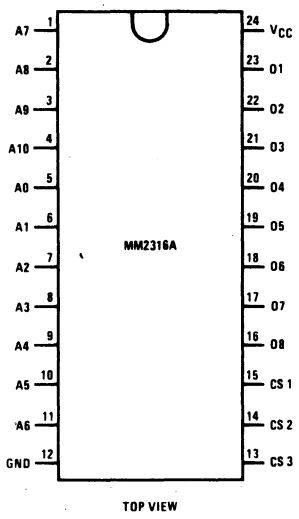
applications

- Microprogramming
- Control logic
- Table look-up

block and connection diagrams

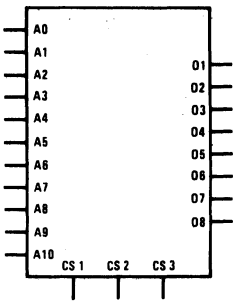


Dual-In-Line Package



TOP VIEW

logic symbol



IM-B20M67/Printed in U.S.A.

absolute maximum ratings (Note 1)

Voltage at Any Pin	−0.5V to +7V
Operating Temperature Range	0°C to +70°C
Storage Temperature Range	−65°C to +150°C
Power Dissipation	1W
Lead Temperature (Soldering, 10 seconds)	300°C

dc electrical characteristics (T_A within operating temperature range, V_{CC} = 5V ±5%, unless otherwise noted).

PARAMETER (Note 3)	CONDITIONS	MIN	TYP (Note 4)	MAX	UNITS
Input Current (I_{LI})	V_{IN} = 0 to V_{CC}			10	μA
Logical "1" Input Voltage (V_{IH})		2.0		V_{CC}+1.0	V
Logical "0" Input Voltage (V_{IL})		−0.5		0.8	V
Logical "1" Output Voltage (V_{OH})	I_{OH} = −100 μA	2.2			V
Logical "0" Output Voltage (V_{OL})	I_{OL} = 2 mA			0.45	V
Output Leakage Current (I_{LOH})	V_{OUT} = 4V, $\overline{CS}$ = 2.2V			10	μA
Output Leakage Current (I_{LOL})	V_{OUT} = 0.45V, $\overline{CS}$ = 2.2V			−20	μA
Power Supply Current (I_{CC1})	All Inputs = 5.25V, Data Output Open		40	98	mA

capacitance

PARAMETER (Note 3)	CONDITIONS	MIN	TYP (Note 4)	MAX	UNITS
Input Capacitance (All Inputs) (C_{IN})	V_{IN} = 0V, T_A = 25°C, f = 1 MHz, (Note 2)			7.5	pF
Output Capacitance (C_{OUT})	V_{OUT} = 0V, T_A = 25°C, f = 1 MHz, (Note 2)			15.0	pF

ac electrical characteristics

(T_A within operating temperature range, V_{CC} = 5V ±5%, unless otherwise specified).
See ac test circuit and switching time waveforms.

PARAMETER (Note 3)	CONDITIONS	MIN	TYP (Note 4)	MAX	UNITS
t_{ACCESS}	See ac Load Circuit. All Times Measured to 1.5V Level with t_r and t_f of Input < 20 ns, *(Figure 1)*			450	ns
t_{SELECT}	See ac Load Circuit. All Times Measured to 1.5V Level with t_r and t_f of Input < 20 ns, *(Figure 2)*			300	ns
$t_{DESELECT}$	See ac Load Circuit. All Times Measured to 1.5V Level with t_r and t_f of Input < 20 ns, *(Figure 2)*			300	ns

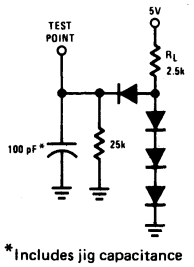
Note 1: "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. Except for "Operating Temperature Range" they are not meant to imply that the devices should be operated at these limits. The table of "Electrical Characteristics" provides conditions for actual device operation.

Note 2: Capacitance is guaranteed by periodic testing.

Note 3: Positive true logic notation is used: logical "1" = most positive voltage level, logical "0" = most negative voltage level.

Note 4: Typical values are for T_A = 25°C and nominal supply voltage.

ac test circuit and switching time waveforms



*Includes jig capacitance

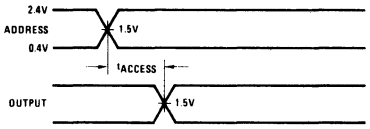


FIGURE 1. Access Time

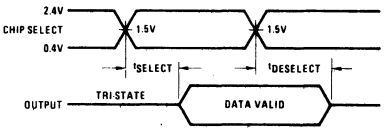


FIGURE 2. Output Enable and Disable

custom ROM programming

Custom ROM programs are submitted to National in three formats: paper tape, punched cards or truth table, with punched cards being the preferred. These programs are converted into machine language and outputted on a magnetic tape. This magnetic tape is used to make the programmable mask and the test tape. The wafers are tested at the wafer level. The wafer is then scribed and the good dice assembled. After assembly, the units are tested using the custom test tape to assure the correct output pattern for every address.

National has programs to convert NEGATIVE logic to POSITIVE or POSITIVE to NEGATIVE so ROMs can be entered in either logic, but the customer must specify which logic definition is used.

PROGRAMMING DEFINITIONS

Logic Definitions

NEGATIVE Logic: "0" = V_H = the more positive voltage. "1" = V_L = the more negative voltage.

POSITIVE Logic: "0" = V_L = the more negative voltage. "1" = V_H = the more positive voltage.

Input/Output Definitions

Address: A0 is the least significant input address.

Outputs: O1 is the least significant output.

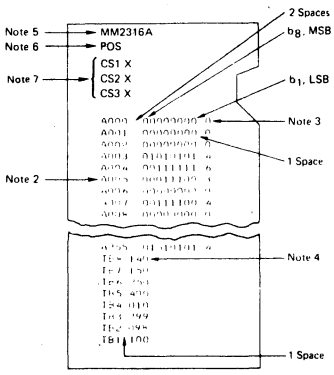
Custom ROM Programming

INFORMATION NEEDED

So that National can better serve its customers, the following information *must* be submitted with each ROM code.

National Semiconductor Corporation 2900 Semiconductor Dr., Santa Clara, CA 95051 Phone (408) 737-5000 TWX 910-339-9240			NATIONAL PART NUMBER
			ROM LETTER CODE (NATIONAL USE ONLY)
NAME			DATE
ADDRESS			CUSTOMER PRINT OR I.D. NO.
CITY		STATE ZIP	PURCHASE ORDER NO.
TELEPHONE	NAME OF PERSON NATIONAL CAN CONTACT (PRINT)	AUTHORIZED SIGNATURE	DATE

tape entry format (Note 1)



MM2316A

8-Bit Tape Format

Note 1: The code is a 7-bit ASCII code on 8 punch tape.

Note 2: The ROM input address is expressed in decimal form and is preceded by the letter A.

Note 3: The total number of "1" bits in the output word.

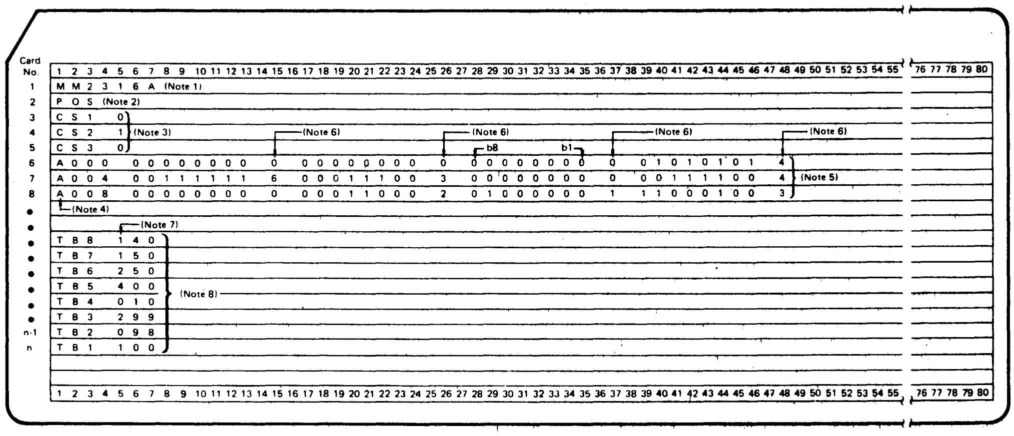
Note 4: The total number of "1" bits in each output column or bit position.

Note 5: Specify product type.

Note 6: Must type POS logic, or NEG logic depending on which is used. Logic on addresses and outputs must be the same (either POS or NEG).

Note 7: Specify the pattern necessary to select the ROM.

card entry format



Note 1: Specify product type.

Note 2: Must type POS logic or NEG logic depending on which is used. Logic on addresses, outputs and chip selects must be the same (either POS or NEG).

Note 3: Specify the chip select logic levels that will enable the ROM.

Note 4: The first ROM input address per card is expressed in decimal form and is preceded by the letter A.

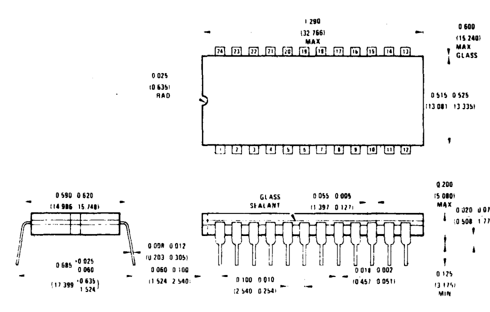
Note 5: Punch four address locations per card, only first location on each card has the address location expressed in decimal form.

Note 6: The total number of "1" bits in the output word.

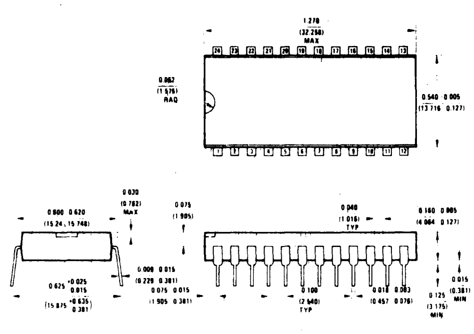
Note 7: Leading zeros must be punched.

Note 8: The total number of "1" bits in each output column or bit position.

physical dimensions



Cavity Dual-In-Line Package (J)
Order Number MM2316AJ
NS Package Number J24A



Molded Dual-In-Line Package (N)
Order Number MM2316AN
NS Package Number N24A

Manufactured under one or more of the following U.S. patents: 3083262, 3189758, 3231797, 3303356, 3317671, 3323071, 3381071, 3408542, 3421025, 3426423, 3440498, 3518750, 3519897, 3557431, 3560765, 3566218, 3571630, 3575609, 3579059, 3593069, 3597640, 3607469, 3617859, 3631312, 3633052, 3638131, 3648071, 3651565, 3693248.

National Semiconductor Corporation
2900 Semiconductor Drive, Santa Clara, California 95051, (408) 737-5000/TWX (910) 339-9240
National Semiconductor GmbH
808 Fuerstenfeldbruck, Industriestrasse 10, West Germany, Tele. (08141) 1371/Telex 05-27649
National Semiconductor (UK) Ltd.
Larkfield Industrial Estate, Greenock, Scotland, Tele. (0475) 33251/Telex 778-632

March 1977
Pub. No. 426305290-0018

ISP-8A/600 single-chip 8-bit n-channel microprocessor (SC/MP-II)

general description

SC/MP (Simple Cost-effective MicroProcessor) is a single-chip 8-bit microprocessor packaged in a standard, 40-pin, dual-in-line package.

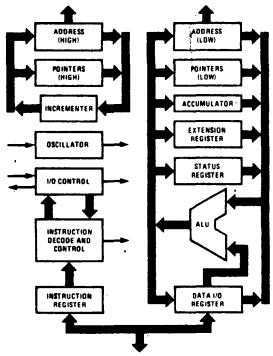
N-channel, silicon gate, depletion mode standard-process technology ensures high performance, high reliability, and high producibility.

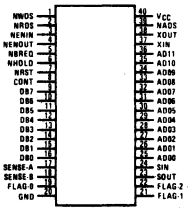
SC/MP is intended for use in general-purpose applications where cost per function is a most significant criterion. But cost efficiency is only a part of SC/MP's story. It goes on to include a variety of useful functions that are not even provided by some of the expensive microprocessors, like self-contained timing circuitry, 16-bit (65k) addressing capability, serial or parallel data-transfer capability and common memory/peripheral instructions. The built-in features in conjunction with the low initial cost describe what SC/MP really is – a microprocessor specifically designed to provide the simplest and most efficient solution to many application requirements.

customer benefits

- Simpler interfacing
  - Bidirectional TRI-STATE® 8-bit data bus
  - TTL-compatible input/output interface
- Si-gate N-channel ion-implant process
- Direct Memory Access (DMA) and multiprocessor capabilities
  - Handshake bus-access control on chip
- Simplified programming
  - Multiple addressing modes – program-counter-relative, immediate data, indexed, auto-indexed, and implied
- Direct control output
  - Three user-accessible control-flag outputs
- Simpler I/O hardware
  - Separate serial-data input and output ports
  - Two sense inputs
  - Direct interfacing to standard memory parts
- Simplified timing hardware
  - On-chip clock generator
- Interface flexibility
  - Capability to interface with memories or peripherals of any speed
- Large system capability
  - Address capability to 65k bytes of memory
- Simplified power requirements
  - Single 5-volt supply
  - Low power
- Lower cost
  - Plastic package

block and connection diagrams





SC/MP-II Pin Configuration

DA-B20M37/Printed in U.S.A.

applications

- Test Systems and Instrumentation
- Machine Tool Control
- Small Business Machines
- Word Processing Systems
- Educational Systems
- Multiprocessor Systems
- Process Controllers
- Terminals
- Traffic Controls
- Laboratory Controllers
- Sophisticated Games
- Automotive

absolute maximum ratings (Note 1)

Voltage at Any Pin . . . . . -0.5V to +7.0V

Operating Temperature Range . . . . . 0°C to +70°C

Storage Temperature Range . . . . . -65°C to +150°C

Lead Temperature (Soldering, 10 seconds) . . . . . 300°C

dc electrical characteristics (T_A = 0°C to +70°C, V_{CC} = +5V ± 5%)

Parameter	Conditions	Min.	Max.	Units
INPUT SPECIFICATIONS				
All Input Pins Except V_{CC} and GND Logic "1" Input Voltage		2.0	V_{CC}	V
Logic "0" Input Voltage		-0.5	0.8	V
Input Capacitance (All pins except V_{CC} and GND)			10	pF
Supply Current I_{CC}	T_A = 25°C outputs unloaded		45	mA
	T_A = 0°C outputs unloaded		50	mA
OUTPUT SPECIFICATIONS				
"TRI-STATE®" Pins (NWDS, NRDS, DB0 - DB7, AD00 - AD11) Logic "1" Output Voltage	I_{OUT} = -100μA	2.4		V
Logic "0" Output Voltage	I_{OUT} = 2.0mA		0.4	V
NADS, FLAG 0 - 2, SOUT, NENOUT Logic "1" Output Voltage	I_{OUT} = -100μA	V_{CC} - 1		V
Logic "1" Output Voltage	I_{OUT} = -1mA	1.5		V
Logic "0" Output Voltage	I_{OUT} = 2.0mA		0.4	V
NBREQ (Note 2) Logic "0" Output Voltage	I_{OUT} = 2.0mA		0.4	V
Logic "1" Output Current	0 ≤ V_{OUT} ≤ V_{CC}		±10	μA
XOUT Logic "1" Output Voltage	I_{OUT} = -100μA	2.4		V
Logic "0" Output Voltage	I_{OUT} = 1.6mA		0.4	V

ac electrical characteristics [T_A = 0°C to +70°C, V_{CC} = +5V ± 5%, 1 TTL Load (Note 3)]

Parameter	Conditions	Min.	Max.	Units
f_x		0.1	4.0	MHz
	R = 240Ω ± 5% (figure 2B) C = 300pF ± 10%	2.0	4.0	MHz
T_C (Note 4)		500		ns
Microcycle		1		μs
External Clock Input (see figure 2A) T_{W0}		120		ns
T_{W1}		120		ns
XOUT/ADS Timing Relationship (see figure 3) T_H (ADS)		100	225	ns
Address and Input/Output Status (see figures 5 and 6) T_{D1} (ADS)			$3T_C/2$	ns
T_W (ADS)		$(T_C/2) - 50$		ns
T_S (ADDR)		$(T_C/2) - 165$		ns
T_H (ADDR)		50		ns
T_S (STAT)		$(T_C/2) - 150$		ns
T_H (STAT)		50		ns
T_H (NBREQ)		0		ns
Data Input Cycle (see figure 5) T_D (RDS)		0		ns
T_W (RDS)		$T_C + 50$		ns
T_S (RD)		175		ns
T_H (RD)		0		ns
T_{ACC} (RD)		$2T_C - 200$		ns
Data Output Cycle (see figure 6) T_D (WDS)		$T_C - 50$		ns
T_W (WDS)		$T_C - 75$		ns
T_S (WD)		$(T_C/2) - 200$		ns
T_H (WD)		100		ns
Input/Output Cycle Extend (see figure 7) T_S (HOLD)		200		ns
T_{D1} (HOLD)		130	275	ns
T_{D2} (HOLD)			350	ns
T_W (HOLD)			∞	ns
T_H (HOLD)		0		ns
Bus Access (see figure 4) T_D (NENOUT)			150	ns
T_{D2} (ADS)		$T_C/2$	$3T_C/2$	ns
T_H (NENIN)		0		ns
Output Load Capacitance XOUT			30	pF
All Other Output Pins			75	pF

Note 1: Maximum ratings indicate limits beyond which damage may occur. Continuous operation at these limits is not intended and should be limited to those conditions specified under electrical characteristics.

Note 2: NBREQ is an input/output signal that requires an external resistor to V_{CC}.

Note 3: All times measured from valid Logic "0" level = 0.8 V or valid Logic "1" level = 2.0 V.

Note 4: T_C is the time period for two clock cycles of the on-chip or external oscillator ($T_C = 2/f_x$). Refer to paragraph titled Timing Control for detailed definition.

Note 5: All times measured with a 50% duty cycle on the external clock.

functional description

SC/MP is a self-contained general-purpose microprocessor designed for ease of implementation in stand alone, DMA (Direct Memory Access), and multiprocessor applications. Communications between SC/MP and external memory peripheral devices are effected via a 12-bit dedicated address bus and an 8-bit bidirectional data bus. During the address interval of each input output cycle, SC/MP employs both busses to provide a 16 bit address output the 12 least significant address bits are sent out over the 12-bit address bus and the 4 most significant address bits are sent out over the 8-bit data bus along with 4 status bits. Separate strobe outputs from SC/MP (NADS, NWDS, NRDS) indicate when valid address information is present on the two busses, and when valid input/output memory or peripheral data are present on the 8 bit bus. To further extend flexibility of application, serial data input-output ports are also provided so that serial data transfers can be effected under program control. The remaining input/output signals shown in figure 1 are dedicated to general-purpose control and status functions, including initialization, bus management, microprocessor halt, interrupt request, input/output cycle extension, and user-specified hardware/software interface functions. A detailed description of each input/output signal is provided in table 1.

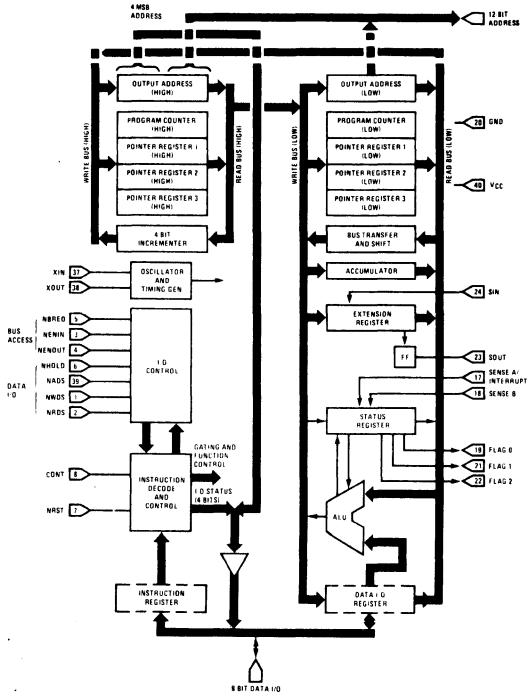


FIGURE 1. SC/MP-II Detailed Block Diagram

TABLE 1. Input/Output Signal Description

Signal Mnemonic	Functional Name	Description
NRST	Reset Input	Set high for normal operation. When set low, aborts in-process operations. When returned high, internal control circuit zeroes all programmer-accessible registers; then, first instruction is fetched from memory location 0001_{16}.
CONT	Continue Input	When set high, enables normal execution of program stored in external memory. When set low, SC/MP operation is suspended (after completion of current instruction) without loss of internal status.
NBREQ	Bus Request Input/Output	Associated with SC/MP internal allocation logic for system bus. Can be used as bus request output or bus busy input. Requires external load resistor to V_{CC}.
NENIN	Enable Input	Associated with SC/MP internal allocation logic for system bus. When set low, SC/MP is granted access to system busses. When set high, places system busses in high-impedance (TRI-STATE®) mode.
NENOUT	Enable Output	Associated with SC/MP internal allocation logic for system bus. Set low when NENIN is low and SC/MP is not using system busses (NBREQ-high). Set high at all other times.
NADS	Address Strobe Output	Active-low strobe. While low, indicates that valid address and status output are present on system busses.
NRDS	Read Strobe Output	Active-low strobe. On trailing edge, data are input to SC/MP from 8-bit bidirectional data bus. High-impedance (TRI-STATE®) output when input/output cycle is not in progress.
NWDS	Write Strobe Output	Active-low strobe. While low, indicates that valid output data are present on 8-bit bidirectional data bus. High-impedance (TRI-STATE®) output when input/output cycle not in progress.
NHOLD	Input/Output Cycle Extend Input	When set low prior to trailing edge of NRDS or NWDS strobe, stretches strobe to extend input/output cycle; that is, strobe is held low until NHOLD signal is returned high.
SENSE A	Sense/Interrupt Request Input	Serves as interrupt request input when SC/MP internal IE (Interrupt Enable) flag is set. When IE flag is reset, serves as user-designated sense condition input. Sense condition testing is effected by copying status register to accumulator.
SENSE B	Sense Input	User designated sense-condition input. Sense condition testing is effected by copying status register to accumulator.
SIN	Serial Input to E Register	Under software control, data on this line are right-shifted into E register by execution of SIO instruction.
SOUT	Serial Output from E Register	Under software control, data are right-shifted onto this line from E register by execution of SIO instruction. Each data bit remains latched until execution of next SIO instruction.
FLAGS 0, 1, 2	Flag Outputs	User-designated general-purpose flag outputs of status register. Under program control, flags can be set and reset by copying accumulator to status register.
AD00-AD11	Address Bit 00 through Address Bit 11	Twelve TRI-STATE® address output lines. SC/MP outputs 12 least significant address bits on this bus when NADS strobe is low. Address bits are then held valid until trailing edge of read (NRDS) or write (NWDS) strobes. After trailing edge of NRDS or NWDS strobe, bus is set to high impedance (TRI-STATE®) mode until next NADS strobe.

NOTE:

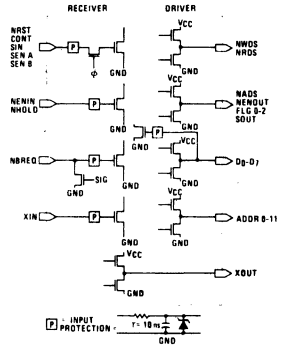
The 8-bit bidirectional data bus is set to the high impedance (TRI-STATE®) mode except when it is actually in use by SC/MP (NADS, NRDS, or NWDS low). During the addressing interval of each input/output cycle (NADS low), SC/MP provides address and status outputs over the bus; during the ensuing data transfer interval (NRDS or NWDS low), 8-bit input or output data bytes are routed over the bus.

TABLE 1. Input/Output Signal Description (Continued)

Signal Mnemonic/ Pin Designation	Input at NADS Time: Functional Name	Input at NADS Time: Description	Input at NRDS Time	Output at NWDS Time
DB0	Address Bit 12	Fourth most significant bit of 16-bit address.	Input data are expected on the eight (DB0-DB7) lines.	Output data are valid on the eight (DB0-DB7) lines.
DB1	Address Bit 13	Third most significant bit of 16-bit address.		
DB2	Address Bit 14	Second most significant bit of 16-bit address.		
DB3	Address Bit 15	Most significant bit of 16-bit address.		
DB4	R-Flag	When high, data input cycle is starting; when low, data output cycle is starting.		
DB5	I-Flag	When high, first byte of instruction is being fetched.		
DB6	D-Flag	When high, indicates delay cycle is starting, that is, second byte of DLY instruction is being fetched.		
DB7	H-Flag	When high, indicates that Halt Instruction has been executed. (In some system configurations, the H-Flag output is latched and, in conjunction with the CONTinue input, provides a programmed halt.)	**Note:** The DB0 through DB7 (AD12-HFLG) lines are a high-impedance (open circuit) load when SC/MP does not have access to the input/output bus.	

DRIVERS AND RECEIVERS

Equivalent circuits for SC/MP drivers and receivers are shown below. All inputs have static charge protection circuits consisting of an RC filter and voltage clamp. These devices still should be handled with care, as the protection circuits can be destroyed by excessive static charge.



SC/MP-II Driver and Receiver Equivalent Circuits

TIMING CONTROL

All necessary timing signals are provided by a three-stage inverter ring oscillator contained on the SC/MP chip. Two control pins, XIN and XOUT, permit the frequency of the oscillator to be controlled by any of the following methods:

1. By leaving the XOUT pin unterminated and driving the XIN pin with an externally generated TTL clock that conforms to the parameters shown in figure 2A. For this method, the frequency of the oscillator is equal to the frequency of the external clock input.

2. By connecting a resistor-capacitor feedback network between the XIN and XOUT pins and GND as shown in figure 2B.

3. By connecting a crystal with low-pass filter network between the XIN and XOUT pins and GND as shown in figure 2C (for above 1 megahertz) or figure 2D (for 1 megahertz or below). For this method, the frequency of the oscillator is equal to the resonant frequency of the crystal and the low-pass filter prevents unwanted harmonic oscillations.

In addition to illustrating appropriate frequency-control networks for the on-chip oscillator, figures 2A through 2D also show how an optional driver may be used to derive a system clock from the oscillator signal present at the XOUT pin. For reference purposes, the timing relationship between the XOUT signal and the NADS strobe is shown in figure 3.

In the discussions that follow, instruction execution and input/output timing are described in terms of microcycles.

The time interval of a microcycle is four times the period of the oscillator; that is:

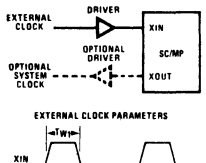
$$\text{period of one microcycle} = 2T_C$$

$$T_C = 2(\frac{1}{f_{osc}}) = 2(\frac{1}{f_{res}}) = 2(\frac{1}{f_{XIN}})$$

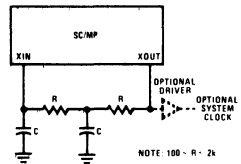
where:

T_C = time period for two cycles of on-chip or external oscillator
f_{osc} = frequency of on-chip oscillator
f_{res} = resonant frequency of crystal connected between XIN and XOUT pins
f_{XIN} = frequency of external clock applied to XIN pin

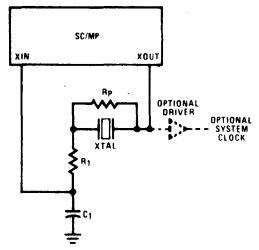
A. External Clock Input



B. Resistor-Capacitor Feedback Network



C. Crystal with Low-Pass Filter (Above 1MHz)

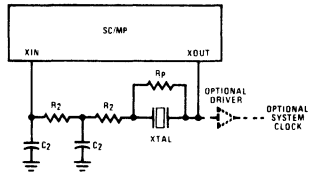


Suggested values for Crystal with Low Pass Filter Network

Crystal	R_P	C_1	R_1
2MHz	100kΩ	56pF	1kΩ
3.58MHz	100kΩ	27pF	1kΩ
4MHz	100kΩ	27pF	1kΩ

XTAL is parallel resonant with maximum series resonance equal to 1kΩ.

D. Crystal with Low Pass Filter (1 MHz or Below)



Typical Oscillator Frequency vs RC Time Constant

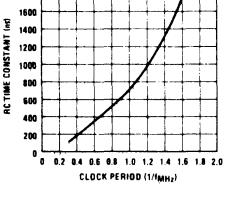


FIGURE 2. Frequency Control Networks for On-Chip Oscillator

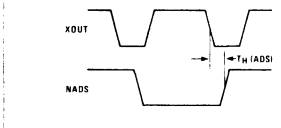


FIGURE 3. XOUT/NADS Timing Relationship

INSTRUCTION FORMAT

The SC/MP instruction repertoire includes both single-byte and double-byte instructions. A single-byte instruction consists of an 8-bit operation code that specifies an operation that SC/MP can execute without further reference to memory. A double-byte instruction consists of an 8-bit operation code and an 8-bit data or displacement field. When the second byte represents a data field, the data are processed by SC/MP during execution of the instruction, thereby eliminating the need for further memory references. When the second byte represents a displacement value, it is used to calculate a memory address that will be accessed (written into or read from) during execution of the instruction (refer to Addressing).

DATA STORAGE

As shown in figure 1, SC/MP provides ten internal registers, seven of which are accessible to the programmer. The purpose and function of these registers are described below:

Program Counter – The program counter is a 16-bit register that contains the address of the instruction being executed. The contents of this register are automatically incremented by one just before each instruction is fetched from memory, to enable sequential execution of the stored instructions. Under program control, the contents of this register also may be modified or exchanged with the contents of a pointer register to effect subroutine calls and program branches.

NOTE:

The 16-bit address output of the program counter consists of a 4-bit high-order address and a 12-bit low-order address. When the program counter is incremented at the start of each instruction fetch input/output cycle, only the 12 low-order bits are affected; no carry is provided to the 4 high-order bits. For systems employing memories of 4k or less, the high-order bits can be ignored as they are set to 0000_{16} following initialization. For systems employing larger memories, the contents of a pointer register can be modified to select the desired 4k block of memory.

Pointer Registers – The pointer registers are 16-bit general-purpose registers that normally are loaded under program control with reference addresses that serve as base pointers, stack pointers, and subroutine pointers. In applications having minimal memory-addressing requirements, these registers may be used alternately as data storage registers.

NOTE:

When interrupt requests are enabled, pointer register 3 is automatically referenced by the internal microprogram for formation of the starting address of the user-generated interrupt service routine. (See figure 9.) In this case, the contents of pointer register 3 must be set to one less than the memory location of the first instruction in the interrupt service routine.

Accumulator – The 8-bit accumulator (AC) is the primary working register of SC/MP. It is used for performing and storing the results of arithmetic and logic operations as well as for data transfers, shifts, rotates, and data exchanges with the program counter, the pointer registers, and the status register.

Extension Register – The extension register is used both for serial input/output data transfers and with the accumulator to effect arithmetic, logic, and data-transfer operations. If the second byte of an indexed or auto-indexed memory-reference instruction (refer to Addressing) equals -128_{10}, the contents of the extension register are used as the displacement value for address formation.

Status Register – The status register provides storage for arithmetic, control, and software status flags. For more-detailed information on the function of this register, refer to Status Register under the description of the Arithmetic and Logic Unit.

Instruction Register – The 8-bit instruction register is not accessible to the programmer. During the fetch phase of each instruction cycle, this register is loaded with the 8-bit instruction operation code retrieved from memory (for a single-byte instruction or the first byte of a double-byte instruction).

Data Input/Output Register – The data input/output register is not accessible to the programmer. It is used for temporary storage of all input/output data received via or transmitted over the 8-bit bidirectional data bus during the data-transfer interval of each input/output cycle (NRDS or NWDS low).

Address Register – The 16-bit address register is not accessible to the programmer. It is used for temporary storage of the 16-bit address transmitted during an input/output cycle.

ARITHMETIC AND LOGIC UNIT

The Arithmetic and Logic Unit (ALU) provides the data-manipulation capability that is an essential feature of any microprocessor. The operations provided by the ALU include OR, XOR, increment, decrement, binary addition, and decimal addition. For decimal addition, the data inputs to the ALU are treated as two 4-bit BCD digits, thereby eliminating the program-storage and execution time required to perform BCD to binary conversion.

BUS TRANSFER LOGIC

The bus transfer logic processes the gating and function control outputs of the instruction-decode logic to provide the shift-right (with link, without link, or with serial input data), rotate (with or without link), and bus-exchange functions necessary for data movement between the SC/MP internal read and write busses. A general summary of the data-manipulation capabilities available to the programmer follows.

1. Either the low-order or the high-order byte of any pointer register can be exchanged with the contents of the 8-bit accumulator. Thus, data exchanges between the pointer registers can be effected one byte at a time via the accumulator.

2. The contents of the program counter can be directly exchanged with the contents of any pointer register.

3. The contents of the extension register can be loaded into the accumulator or can be exchanged with the contents of the accumulator. When the accumulator is loaded from the extension register, the original contents of the accumulator are lost.

4. The contents of the status register can be copied into the accumulator to enable status modification or conditional branch testing. When the status register is copied into the accumulator, the contents of the status register are not altered but the original contents of the accumulator are lost.

5. The contents of the accumulator can be copied into the status register to change the outputs of the status register, except for status bits 4 and 5 (Sense A and B inputs to SC/MP). Since these are read-only bits, they are not affected by data movements internal to SC/MP. Copying the accumulator into the status register does not alter the contents of the accumulator.

NOTE:

The flag 0, 1, and 2 outputs of the status register serve as latched flags; in other words, they are set to the specified state when the contents of the accumulator are copied into the status register, and they remain in the specified state until the contents of the status register are modified again under program control.

STATUS REGISTER

The function of each bit in the status register is described briefly below.

7	6	5	4	3	2	1	0
CY/L	OV	S_B	S_A	IE	F_2	F_1	F_0

User Flag 0 – User-assigned general-purpose status bit for implementation as software status bit or in system control applications. This status bit is available as an external output from SC/MP.

User Flag 1 – Same as User Flag 0.

User Flag 2 – Same as User Flag 0.

Interrupt Enable Flag – Internal status bit that is set and reset under program control. When set, SC/MP recognizes external interrupt requests received via Sense A input. When reset, inhibits SC/MP from recognizing interrupt requests.

Sense A – General-purpose status input for sensing external conditions. When IE flag is reset, this bit can be tested by copying status register to accumulator. When IE flag is set, this bit serves as interrupt request input causing SC/MP to automatically branch to user-generated interrupt-service routine in response to high input.

Sense B – Same as Sense A except that it is not tested for interrupt status.

NOTE:

Sense A and B inputs are read-only bits. Thus, they are not affected when the contents of the accumulator are copied into the status register.

Overflow (OV) – This bit is set if an arithmetic overflow occurs during an add (ADD, ADI, or ADE) or a complement-and-add instruction (CAD, CAI, or CAE). It is not affected by the decimal-add instructions (DAD, DAI, or DAE).

Carry/Link (CY/L) – This bit is set if a carry from the most significant bit occurs during an add, complement-and-add, or decimal-add instruction. Thus, it serves as a carry input to the next add instruction. In addition, it is included in the Shift Right with Link (SRL) and Rotate Right with Link (RRL) instructions.

CONTROL

The operation of the SC/MP microprocessor consists of repeatedly accessing or fetching instructions from the program stored in external memory and executing the operations specified by the instructions. These two steps are carried out under the control of an internal microprogram. (SC/MP is not user-microprogrammable.) The microprogram is similar to a state table specifying the series of states of system control signals necessary to carry out each instruction. Microprogram storage is provided in the instruction decode and control logic, and microprogram routines are implemented to fetch and execute instructions. The fetch routine first increments the program counter, and then causes the instruction address to be transferred from the program counter to the system busses via the output address register. The microprogram next initiates an input data transfer. When the instruction operation code is subsequently placed on the 8-bit data bus (single-byte instruction or first byte of double-byte instruction), the operation code is loaded into the instruction register. The operation code is then partially decoded to determine whether the instruction contains a second byte. If it does, a second input data transfer is effected to load the next byte in the data input/output register.

After the complete instruction is stored in the instruction and/or data input/output register(s), the instruction decoder transforms the instruction operation code into the address of the appropriate instruction-execution routine contained in the internal microprogram. The microprogram then branches to the specified internal address to initiate execution of the instruction. The resulting execution routine comprises one or more microinstructions that implement the required functions. For example, the first microcycle of an Extension Register Add Instruction (ADE) causes the contents of the extension register to be gated onto the read bus, transferred to the write bus via the bus control logic, and then written into the data input/output register. The next microcycle causes the contents of the accumulator to be gated onto the read bus, the contents of the read bus to be added to the contents of the data input/output register via the ALU, and the resultant output of the ALU to be written into the accumulator via the write bus. The final step of the execution routine is a jump back to the fetch routine to access the next instruction.

INITIALIZATION

Since SC/MP may power up in a random condition, the following power-up and initialization procedure is recommended.

1. Apply power (GND and V_{CC}) and set NRST low.

NOTE:

Allow ample time (typically, 250 ms) for the oscillator and the internal clocks to stabilize. In systems where NRST is set low after turning on power, NRST must remain low for a minimum of $4T_C$. While NRST is low, any in-process operations are aborted automatically. When NRST is low, strobes and address and data busses are in the Non-I/O state (high-Z state).

2. Set NRST high. If the rise time of this input is too slow, the processor, first, will initialize and execute a few instructions and, then, will reinitialize. If the application is such that multiple initialization is undesireable, NRST should be brought high at a minimum rate of 2 volts per microcycle.

NOTE:

This causes the SC/MP internal control circuit to set the contents of all programmer-accessible registers to zero. Thus, when SC/MP is granted access to the system busses following initialization, the first instruction is fetched *always* from memory location 0001_{16}. The NBREQ output goes low, indicating the start of this input/output cycle; this occurs at a time within $13T_C$ after NRST is set high. Normal execution of the program continues as long as NRST remains high.

parallel data transfers

Parallel data transfers occur during each instruction fetch and during the ensuing read/write cycle associated with execution of the memory-reference instructions. This class of instruction could perhaps more properly be called the "Input/Output Reference Class" in the case of the SC/MP microprocessor, since all data transfers, whether with memory, peripheral devices, or a central processor data bus, occur through the execution of these instructions. This unified bus structure is in contrast with many other microprocessors and minicomputers that have one instruction type (input/output class) for communication with peripheral devices and another instruction type (memory reference class) for communication with memories. The advantage of the approach taken by SC/MP is that a wider variety of instructions (the entire memory-reference class) is available for communications with peripherals. Thus, the LD and ST (Load and Store) instructions can be used for basic transfers, the ILD and DLD (increment/decrement and load) instructions can be used for indexing peripheral registers, and the remaining memory reference instructions can be used, as required, for "one-step" retrieval and processing of peripheral input data.

BUS UTILIZATION

The bus utilization of SC/MP is shown in table 2.

NBREQ, NENIN, and NENOUT are active and bus access is controlled as shown in figure 4. If NENIN is returned high during an input/output cycle, the input/output cycle is repeated when NENIN is again returned low.

During an ILD or DLD instruction, SC/MP does not relinquish the bus between the loading of the data and the storing of the modified data. If NENIN is brought high after the data have been loaded, the load portion of the cycle is not repeated when NENIN is returned low.

BUS ACCESS

Before SC/MP can initiate parallel data transfers with memory or peripheral devices, it must have access to the system address and data busses. Three of the SC/MP input/output signals are associated with bus control: NBREQ, NENIN, and NENOUT. For simple stand-alone applications, the NENOUT signal can be ignored and the NENIN signal can be tied to GND to allow the SC/MP microprocessor to have continual access to the system busses. The NBREQ input/output line then goes low during each input/output cycle as shown in figures 5 and 6 to indicate when SC/MP is actually using the system busses.

NOTE:

The NBREQ input/output line must be tied to V_{CC} via an external load resistor to allow normal operation of the SC/MP microprocessor.

For DMA and multiprocessor applications, the NBREQ, NENIN, and NENOUT signals can be interconnected in various configurations to allow bus access to be granted to requesting devices according to user-specified priorities. Figure 4 illustrates the general sequence in which these signals are processed by SC/MP to gain access to the system busses and to indicate when the busses are actually being used.

INPUT/OUTPUT CYCLE

Once SC/MP has control of the system busses, the actual input/output cycle begins. As shown in figures 5 and 6, the functions of memory addressing, data reading, and data writing are implemented, respectively, by the address strobe (NADS), the read strobe (NRDS), and the write strobe (NWDS). Note that the NBREQ signal is reset high at the end of the input/output cycle to indicate that the system busses are now free for use by the highest-priority requesting device.

The first operation that SC/MP performs for each input/output cycle is to load the 12 least significant address bits onto the 12-bit address bus, and the 4 most significant address bits along with 4 status bits onto the 8-bit data bus. At the same time, SC/MP sets the NADS output low to indicate that the address and the status information are valid. The low-order address on the 12-bit bus is then held valid for the duration of the input/output cycle; the high-order address and the status information on the 8-bit bus remain valid only while NADS is low. While valid, the status bits have the following significance:

RFLG – When high, indicates that input/output cycle is read cycle; when low, indicates that input/output cycle is write cycle.

IFLG – Set high to indicate that instruction operation code (single-byte instruction or first byte of double-byte instruction) will be output from memory following NADS.

DFLG – Set high only when second byte of Delay Instruction is to be read from memory following NADS. Execution of the Delay Instruction then starts at trailing edge of NRDS. Upon completion, SC/MP provides NADS output to initiate next input/output cycle if bus access is granted. Time in microcycles from leading edge of delay flag to leading edge of subsequent NADS output is computed from the following formula:

$$\text{Delay} = [9 + 2(\text{AC}) + 2\ \text{disp} + 2^9\ \text{disp}]\ \text{microcycles}$$

where

(AC) = unsigned contents of accumulator

disp = unsigned displacement value contained in second byte of Delay Instruction

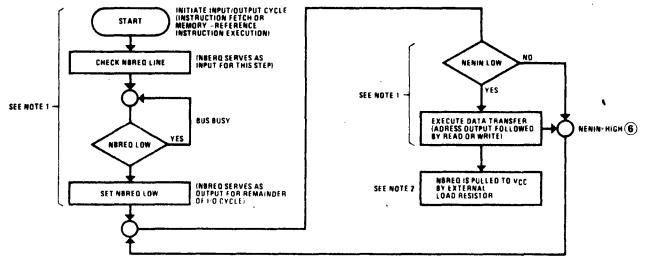
The time derived from the above formula does not include the four microcycles required to fetch the first byte of the Delay Instruction. Thus, when the Delay

Instruction is used for software timing, total instruction execution time equals $[13 + 2(\text{AC}) + 2\ \text{disp} + 2^9\ \text{disp}]$ microcycles.

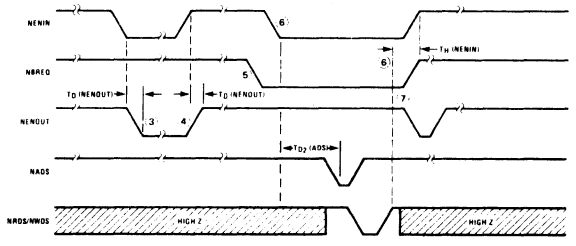
NOTE:

When Halt Instruction is executed, instruction decode and control logic inhibits incrementing of program counter for one input/output cycle. Thus, Halt Instruction is read from memory a second time to enable generation of HFLG output, but no further processing of Halt Instruction occurs. In effect, this procedure ensures HFLG is output in advance of the next instruction to be fetched from memory.

HFLG – Set high only during addressing interval of read cycle that follows Halt Instruction. HFLG may be used to cause user-provided external logic to set the CONT input low, and thereby to effect a programmed halt. Since HFLG read cycle precedes the next instruction



A. NBREQ and NENIN Processing Sequence



B. NBREQ, NENIN, and NENOUT Timing

Note 1: NENOUT is always high while SC/MP is actually using bus, that is, NENIN input and NBREQ output are low.

Note 2: When SC/MP is not using bus (NBREQ output or NENIN input high), NENOUT is held in same state as NENIN input.

Note 3: NENOUT goes low to indicate that SC/MP was granted access to bus (NENIN low) but is not using bus.

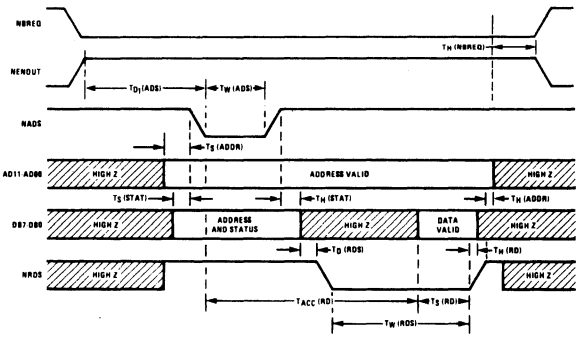
Note 4: NENOUT goes high in response to high NENIN input.

Note 5: SC/MP generates bus request; bus access not granted because NENIN high.

Note 6: NENIN goes low. Bus access now granted and input/output cycle actually initiated. If NENIN is set high while SC/MP has access to the bus, the address and data ports will go to the high-impedance ("Tri State") state, but NBREQ will remain low. When NENIN is subsequently set low, the input/output cycle will begin again.

Note 7: Input/output cycle completed. NENOUT goes low to indicate that SC/MP granted access to bus but not using bus. If NENIN had been set high before completion of input/output cycle, NENOUT would have remained high.

FIGURE 4. Bus Access Control



Note: Timing is valid when NENIN is low before NBREQ is set low by SC/MP; see figure 4 for NADS timing when NENIN is set low after NBREQ.

FIGURE 5. SC/MP Data Input Timing

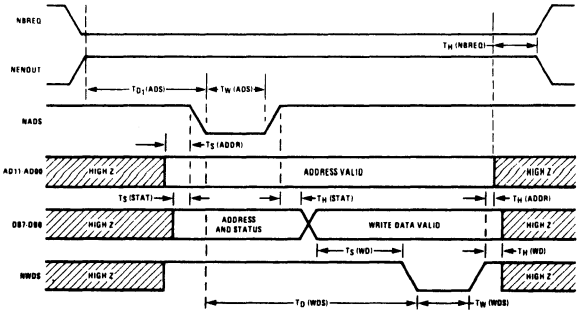


FIGURE 6. Data Output Timing

fetch, termination of programmed halt enables fetch of first instruction that follows Halt Instruction.

After resetting the NADS output, SC/MP generates an NRDS or NWDS strobe, respectively, to initiate a data-input (read) or data-output (write) operation. For a read operation, input data are strobed into SC/MP from the 8-bit bus on the trailing edge of the NRDS strobe. For a write operation, SC/MP places valid output data on the 8-bit bus on the leading edge of the NWDS strobe. After resetting the NRDS or NWDS strobe to complete the data transfer, SC/MP then resets the NBREQ signal to indicate that the system busses are free for use by another controller.

INPUT/OUTPUT CYCLE EXTENSION

As shown in figure 7, the NHOLD signal may be set low prior to the trailing edge of the NRDS or NWDS strobe to cause SC/MP to lengthen the input/output cycle by holding the strobe active until after the NHOLD signal is returned high. Since there is no restriction on the maximum duration of the NHOLD signal, it can be used in a variety of applications ranging from accommodation of memories/peripherals with long access times to single-cycle control of the operating program for software debug purposes.

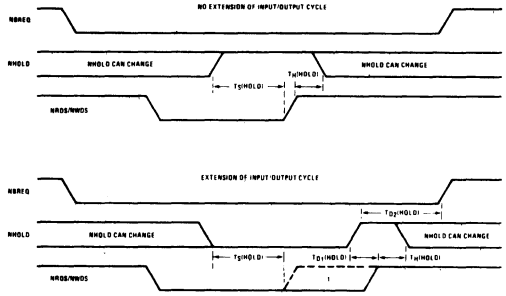
Figure 8 illustrates a typical circuit that may be used to generate an NHOLD signal of repeatable duration. The circuit shown employs a DM74165 8-Bit Parallel In/Serial Out Shift Register to allow selection of an input/output cycle extend time that ranges from $T_C/2$ to $2T_C$ in increments of $T_C/2$. Functional operation of the circuit is controlled by the NADS strobe and XOUT signals. Each time that the NADS strobe goes low, the data present at the A through H terminals are loaded into the shift register in parallel. When the NADS strobe subsequently returns high, the data are then shifted out serially on the positive-to-negative transitions of XOUT.

Thus, the NHOLD output of the circuit is set low on the leading edge of each NADS strobe and, as shown in the chart that accompanies the circuit diagram, it remains low for a time period ranging from three clock cycles minimum (B, C, D, and E inputs = Logic "1") to seven clock cycles maximum (B, C, D, and E inputs = Logic "0").

It is important to note that instruction execution time is increased whenever an input/output cycle is extended via the NHOLD signal. For purposes of computing the increase in instruction execution time, it is necessary to distinguish between the terms *Input/Output Cycle Delay Period* and *Input/Output Cycle Extend Time*. The term *Input/Output Cycle Delay Period* refers to the time that the NRDS/NWDS strobe is actually "stretched" to provide the required memory or peripheral access time. The term *Input/Output Cycle Extend Time* refers to the additional number of microcycles required by the internal SC/MP microprogram to complete the extended input/output cycle; that is:

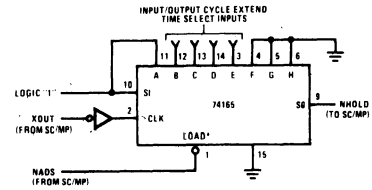
Input/Output Cycle Delay Period	Input/Output Cycle Extend Time
$T_C/2$ through $2T_C$ ($> 0 \leqslant 1$ μcycle)	1 μcycle
$5T_C/2$ through $4T_C$ ($> 1 \leqslant 2$ μcycles)	2 μcycles
$9T_C/2$ through $6T_C$ ($> 2 \leqslant 3$ μcycles)	3 μcycles
etc.	etc.

The total increase in instruction execution time, therefore, is equal to the *Input/Output Cycle Extend Time* multiplied by the total number of input/output cycles associated with the instruction. For example, a DLD Instruction is normally executed in 22 microcycles. Since this instruction employs three read input/output cycles and one write input/output cycle, an *Input/Output Cycle Extend Time* of one microcycle would increase total DLD Instruction execution time to 26 microcycles.



Note 1: In order to extend the input/output cycle, NHOLD must remain low until the point where NRDS/NWDS would have made a low-to-high transition with NHOLD inactive. Dashed line indicates the trailing edge of NRDS/NWDS when NHOLD is not active.

FIGURE 7. NHOLD Timing



Data Inputs B C D E	NHOLD Duration (in clock cycles)	Input/Output Cycle: Delay Period	Input/Output Cycle: Extend Time in Microcycles	Required Memory Access Time (T_{ACC} (RD))
1 1 1 1	3	0	0	$2T_C - 200$
1 1 1 0	4	$T_C/2$	1	$(5T_C/2) - 200$
1 1 0 0	5	T_C	1	$3T_C - 200$
1 0 0 0	6	$3T_C/2$	1	$(7T_C/2) - 200$
0 0 0 0	7	$2T_C$	1	$4T_C - 200$

FIGURE 8. Typical NHOLD Control Circuit

serial data transfers

Serial input/output data transfers can be used efficiently with very slow input/output peripherals such as X Y plotters, teletypewriters, slow-speed printers, and so forth. Such transfers can be effected in any of the following manners.

1. By assigning serial input/output functions to the extension register via the SIO (Serial Input/Output) Instruction. When this instruction is executed, the contents of the extension register are shifted right one bit. At the same time, data present on the SIN line are shifted into bit position 7 of the extension register and the original contents of bit position 0 are shifted into a flip-flop to provide a latched output of the SOUT line. The SOUT data are then held latched until the next SIO instruction is executed.

2. By using one of the status flags as an output data bit and one of the sense lines as an input data bit.

3. By implementing external logic such that only one line of the 8-bit data input output bus is used.

For synchronous systems, serial data input output timing may be provided by program loops that employ the delay instruction, or by using one or more of the transfer instructions (see table 2) to test the output of an external timing circuit. For asynchronous systems, one of the sense inputs can be used for testing bit received/ready status and a pulsed flag output can be provided, under program control, for peripheral indexing each time that a data bit is actually shifted in or out.

Systems that have several input/output devices must be multiplexed; device selection can then be accomplished using the status flag outputs of SC/MP, or by using parallel input/output commands to load an external latch. Systems that do not require serial input/output capability can employ the SIN and SOUT lines as a sense input and flag output, respectively.

interrupts

When the internal interrupt enable (IE) flag is set under program control, the Sense A line is enabled to serve as an interrupt request input; when the IE flag is reset, SC/MP is inhibited from detecting interrupts. Thus, while the IE flag is set, the Sense A input is tested prior to the fetch phase of each instruction as shown in figure 9. Upon detection of an interrupt request (Sense A high), the following events occur automatically.

1. The status register IE flag is reset to prevent SC/MP from responding to any further interrupt requests. Interrupt request capability can then be reenabled during or at the end of the ensuing user-generated interrupt service routine via the IEN (Enable Interrupt) instruction or by copying the accumulator into the status register.

2. The contents of the program counter are exchanged with the contents of the pointer register 3.

3. The contents of the program counter are incremented by one to address the first instruction of the user-generated interrupt service routine.

The interrupt system must be armed before interrupts are enabled. This is accomplished as follows:

1. First, the Interrupt Enable Bit in the Status Register is set true by executing either an Enable Interrupt Instruction (IEN) or a Copy Accumulator to Status Register Instruction (CAS).

2. Second, one additional instruction is fetched and executed.

A return from interrupt is accomplished by executing two instructions: Enable Interrupt (IEN) immediately followed by Exchange Pointer 3 with Program Counter (XPPC 3).

microprocessor halt

The CONT input to SC/MP is provided to enable suspension of operation without loss of internal status. Processing of the CONT input is shown in figure 9. Since this is an asynchronous input, it can be controlled by external timing logic, or as stated previously, the HALT flag output that appears on the 8-bit data bus (during the read cycle that follows execution of a Halt Instruction) can be used with an external circuit to effect a programmed halt condition. Note that when an interrupt request is detected while the CONT input is low, the first instruction of the user-generated interrupt service routine is automatically executed. Thus, the first instruction of the interrupt service routine can be used to reset the external CONT input logic and, thereby, to terminate the microprocessor halt condition if so desired.

After execution of an instruction, the CONT input must be high for a minimum time of $2T_C$ (1 microcycle) in order to fetch and execute the next instruction.

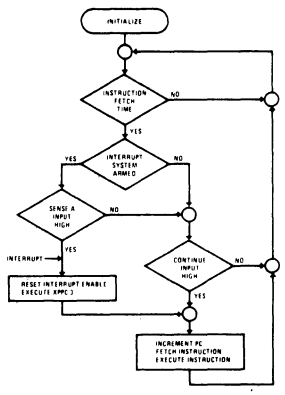


FIGURE 9.

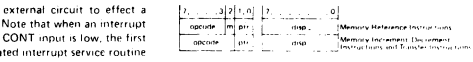
Microprocessor Halt and Interrupt Request Input Processing

instruction set

The SC/MP instruction set provides the general purpose user of microprocessors a powerful programming capability along with above average flexibility and speed. The instruction set consists of 46 instructions, which comprise eight general categories. A listing of the complete instruction set is provided in table 2; typical instruction execution times are given in table 3, and notations and symbols used as shorthand expressions of instruction capability are defined in table 4.

ADDRESSING

During execution, instructions and data defined in a program are stored into and loaded from specific memory locations, the accumulator, or selected registers. Because SC/MP, memory (read/write and read-only), and peripherals are on a common data bus, any instruction used to address memory may be used to address the peripherals. The formats of the instruction groups that reference memory are shown below.



Memory-reference instructions use the PC-relative, indexed, or auto-indexed methods of addressing memory. The memory-increment/decrement instructions and the transfer instructions use the PC-relative or indexed methods of addressing.

The various methods of addressing memory and peripherals are shown below.

Immediate addressing is an addressing format specific to the immediate instruction group.

Type of Addressing	Operand Formats: m	ptr	disp
PC-relative	0	0	128 to +127
Indexed	0	1, 2, or 3	-128 to +127
Immediate	1	0	-128 to +127
Auto-indexed	1	1, 2, or 3	-128 to +127

For PC-relative, indexed, and auto-indexed memory reference instructions, another feature of the addressing architecture is that the contents of the extension register are substituted for the displacement if the instruction displacement equals -128 (-X'80).

NOTE:

All arithmetic operations associated with address formation affect only the 12 low-order address bits; no carry is provided to the 4 high-order bits. For systems employing memories of 4k or less, the high-order bits can be ignored as they are set to 0000 following initialization. For systems employing larger memories, the high order bits must be set to the starting address of the desired 4k block of memory. For example:

0001_2 enables memory locations 1000_{16} - $1FFF_{16}$ to be addressed.

0010_2 enables memory locations 2000_{16} - $2FFF_{16}$ to be addressed and so forth.

PC-Relative Addressing – A PC-relative address is formed by adding the displacement value specified in the operand field of the instruction to the current contents of the program counter. The displacement is an 8 bit twos-

complement number, so the range of the PC-relative addressing format is -128_{10} to $+127_{10}$ locations from the current contents of the program counter.

Immediate Addressing – Immediate addressing uses the value in the second byte of a double-byte instruction as the operand for the operation to be performed (see below).

For example, compare a Load (LD) instruction to a Load Immediate (LDI) instruction. The Load instruction uses the contents of the second byte of the instruction in computing the effective address of the data to be loaded. The Load Immediate instruction uses the contents of the second byte as the data to be loaded.

Indexed Addressing – Indexed addressing enables the programmer to address any location in memory through the use of the pointer register and the displacement. When indexed addressing is specified in an instruction, the contents of the designated pointer register are added to the displacement to form the effective address. The contents of the pointer register are not modified by indexed addressing.

Auto-Indexed Addressing – Auto-indexed addressing provides the same capabilities as indexed addressing along with the ability to increment or decrement the designated pointer register by the value of the displacement. If the displacement is less than zero, the pointer register is decremented by the displacement before the contents of the effective address are fetched or stored. If the displacement is equal to or greater than zero, the pointer register is used as the effective address, and the pointer register is incremented by the displacement after the contents of the effective address are fetched or stored.

system implementation

Figures 10 through 12 illustrate typical SC/MP system configurations. In figure 10, SC/MP is shown interconnected to three memory devices to form a stand-alone 4-device system that provides 256 words of read/write memory and 2,048 words for program storage. Figure 11 shows SC/MP interconnected to an external controller for Direct Memory Access (DMA) operation, and figure 12 illustrates a multiprocessor application using SC/MP's built-in logic to control bus access.

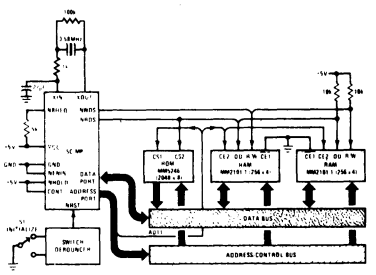


FIGURE 10. SC/MP-II Four-Chip System

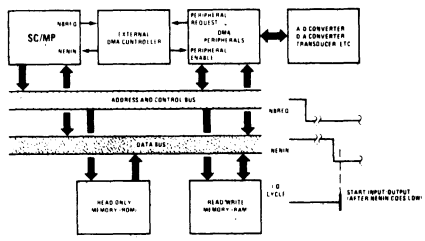


FIGURE 11. SC/MP-II Interconnected for Direct Memory Access (DMA) Operation

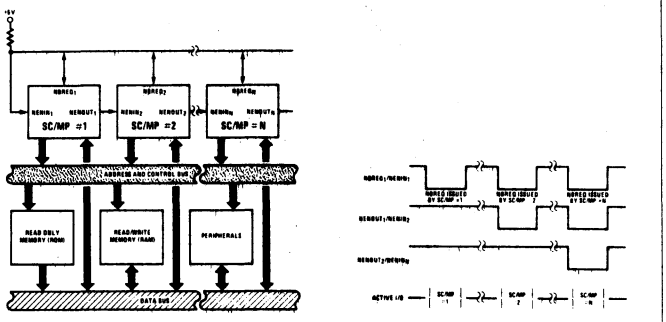


FIGURE 12. Multiprocessor System Using SC/MP-II Built-in Logic for Bus Control

TABLE 2. Bus Utilization of Each Instruction

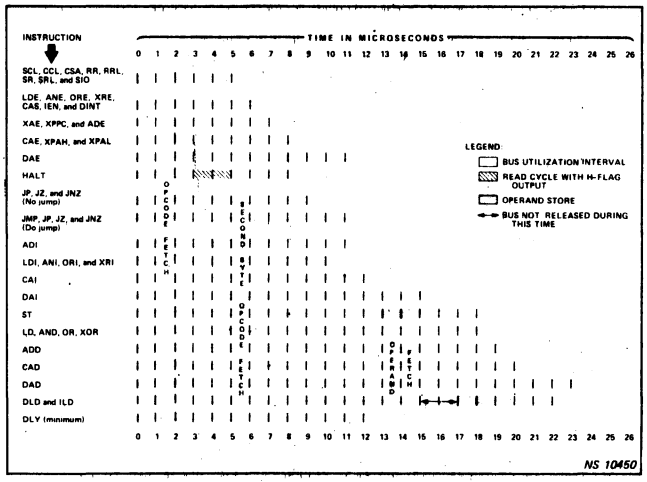


TABLE 3. SC/MP Instruction Summary

DOUBLE-BYTE INSTRUCTIONS

MNEMONIC	DESCRIPTION	OBJECT FORMAT	OPERATION	MICRO-CYCLES
	Memory Reference Instructions	7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0		
LD	Load	1 1 0 0 0 ptr / disp	(AC) ← (EA)	18
ST	Store	1 1 0 0 1	(EA) ← (AC)	18
AND	AND	1 1 0 1 0	(AC) ← (AC) ∧ (EA)	18
OR	OR	1 1 0 1 1	(AC) ← (AC) ∨ (EA)	18
XOR	Exclusive OR	1 1 1 0 0	(AC) ← (AC) ⊻ (EA)	18
DAD	Decimal Add	1 1 1 0 1	(AC) ← (AC)$_{10}$ + (EA)$_{10}$ + (CY/L);(CY/L)	23
ADD	Add	1 1 1 1 0	(AC) ← (AC) + (EA) + (CY/L);(CY/L),(OV)	19
CAD	Complement and Add	1 1 1 1 1	(AC) ← (AC) + ~(EA) + (CY/L);(CY/L),(OV)	20
	Memory Increment/Decrement Instructions	7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0		
ILD	Increment and Load	1 0 1 0 0 ptr / disp	(AC), (EA) ← (EA) + 1	22
DLD	Decrement and Load	1 0 1 1 0	(AC), (EA) ← (EA) - 1	22
	Immediate Instructions	7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0		
LDI	Load Immediate	1 1 0 0 0 1 0 0 / data	(AC) ← data	10
ANI	AND Immediate	1 1 0 1 0 1 0 0 / data	(AC) ← (AC) ∧ data	10
ORI	OR Immediate	1 1 0 1 1 1 0 0 / data	(AC) ← (AC) ∨ data	10
XRI	Exclusive OR Immediate	1 1 1 0 0 1 0 0 / data	(AC) ← (AC) ⊻ data	10
DAI	Decimal Add Immediate	1 1 1 0 1 1 0 0 / data	(AC) ← (AC)$_{10}$ + data$_{10}$ + (CY-L);(CY/L)	16
ADI	Add Immediate	1 1 1 1 0 1 0 0 / data	(AC) ← (AC) + data + (CY-L);(CY/L),(OV)	11
CAI	Complement and Add Immediate	1 1 1 1 1 1 0 0 / data	(AC) ← (AC) + ~data + (CY-L);(CY/L),(OV)	12
	Transfer Instructions	7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0		
JMP	Jump	1 0 0 1 0 0 ptr / disp	(PC) ← EA	11
JP	Jump if Positive	1 0 0 1 0 1	If (AC) > 0, (PC) ← EA	9, 11
JZ	Jump if Zero	1 0 0 1 1 0	If (AC) = 0, (PC) ← EA	9, 11
JNZ	Jump if Not Zero	1 0 0 1 1 1	If (AC) ≠ 0, (PC) ← EA	9, 11
	Double-Byte Miscellaneous Instructions	7 6 5 4 3 2 1 0 / 7 6 5 4 3 2 1 0		
DLY	Delay	1 0 0 0 1 1 1 1 / disp	count AC to 0. delay = 13 + 2(AC) + 2 disp + 2^9 disp microcycles	13 to 131,593

SINGLE-BYTE INSTRUCTIONS

MNEMONIC	DESCRIPTION	OBJECT FORMAT	OPERATION	MICRO-CYCLES
	Extension Register Instructions	7 6 5 4 3 2 1 0		
LDE	Load AC from Extension	0 1 0 0 0 0 0 0	(AC) ← (E)	6
XAE	Exchange AC and Extension	0 0 0 0 0 0 0 1	(AC) ↔ (E)	7
ANE	AND Extension	0 1 0 1 0 0 0 0	(AC) ← (AC) ∧ (E)	6
ORE	OR Extension	0 1 0 1 1 0 0 0	(AC) ← (AC) ∨ (E)	6
XRE	Exclusive OR Extension	0 1 1 0 0 0 0 0	(AC) ← (AC) ⊻ (E)	6
DAE	Decimal Add Extension	0 1 1 0 1 0 0 0	(AC) ← (AC)$_{10}$ + (E)$_{10}$ + (CY-L);(CY/L)	11
ADE	Add Extension	0 1 1 1 0 0 0 0	(AC) ← (AC) + (E) + (CY-L);(CY/L),(OV)	7
CAE	Complement and Add Extension	0 1 1 1 1 0 0 0	(AC) ← (AC) + ~(E) + (CY-L);(CY/L),(OV)	8
	Pointer Register Move Instructions	7 6 5 4 3 2 1 0		
XPAL	Exchange Pointer Low	0 0 1 1 0 0 ptr	(AC) ↔ (PTR$_{7-0}$)	8
XPAH	Exchange Pointer High	0 0 1 1 0 1	(AC) ↔ (PTR$_{15-8}$)	8
XPPC	Exchange Pointer with PC	0 0 1 1 1 1	(PC) ↔ (PTR)	7
	Shift, Rotate, Serial I/O Instructions	7 6 5 4 3 2 1 0		
SIO	Serial Input Output	0 0 0 1 1 0 0 1	(E_0) ← (E_1); SIN → (E_7); (E_0) → SOUT	5
SR	Shift Right	0 0 0 1 1 1 0 0	(AC_7) ← 0; (AC_0) → (CY/L)	5
SRL	Shift Right with Link	0 0 0 1 1 1 0 1	(AC_7) ← (CY/L); (AC_0) → (CY/L)	5
RR	Rotate Right	0 0 0 1 1 1 1 0	(AC_7) ← (AC_0); (AC_0) → (AC_7)	5
RRL	Rotate Right with Link	0 0 0 1 1 1 1 1	(AC_7) ← (CY/L); (AC_0) → (CY/L) → (AC_7)	5
	Single-Byte Miscellaneous Instructions	7 6 5 4 3 2 1 0		
HALT	Halt	0 0 0 0 0 0 0 0	Pulse H flag	8
CCL	Clear Carry/Link	0 0 0 0 0 0 1 0	(CY/L) ← 0	5
SCL	Set Carry/Link	0 0 0 0 0 0 1 1	(CY/L) ← 1	5
DINT	Disable Interrupt	0 0 0 0 0 1 0 0	(IE) ← 0	6
IEN	Enable Interrupt	0 0 0 0 0 1 0 1	(IE) ← 1	6
CSA	Copy Status to AC	0 0 0 0 0 1 1 0	(AC) ← (SR)	5
CAS	Copy AC to Status	0 0 0 0 0 1 1 1	(SR) ← (AC)	6
NOP	No Operation	0 0 0 0 1 0 0 0	None	5

TABLE 4. Instruction Execution Time

INSTRUCTION	READ CYCLES	WRITE CYCLES	TOTAL MICROCYCLES	INSTRUCTION	READ CYCLES	WRITE CYCLES	MICROCYCLES
ADD	3	0	19	JP	2	0	9, 11 for Jump
ADE	1	0	7	JZ	2	0	9, 11 for Jump
ADI	2	0	11	LD	3	0	18
AND	3	0	18	LDE	1	0	6
ANE	1	0	6	LDI	2	0	10
ANI	2	0	10	NOP	1	0	5
CAD	3	0	20	OR	3	0	18
CAE	1	0	8	ORE	1	0	6
CAI	2	0	12	ORI	2	0	10
CAS	1	0	6	RR	1	0	5
CCL	1	0	5	RRL	1	0	5
CSA	1	0	5	SCL	1	0	5
DAD	3	0	23	SIO	1	0	5
DAE	1	0	11	SR	1	0	5
DAI	2	0	15	SRL	1	0	5
DINT	1	0	6	ST	2	1	18
DLD	3	1	22	XAE	1	0	7
DLY	2	0	13 to 131593	XOR	3	0	18
HALT	2	0	8	XPAH	1	0	8
IEN	1	0	6	XPAL	1	0	8
ILD	3	1	22	XPPC	1	0	7
JMP	2	0	11	XRE	1	0	6
JNZ	2	0	9, 11 for Jump	XRI	2	0	10

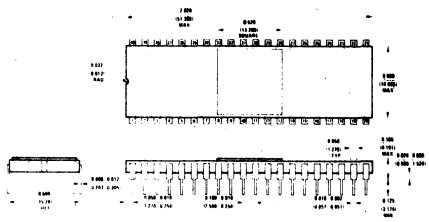
Note: If slow memory is being used, the appropriate delay should be added for each read or write cycle.

TABLE 5. Symbols and Notations Used to Express Instruction Execution

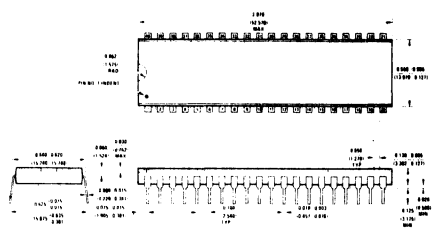
SYMBOL AND NOTATION	MEANING
AC	8-bit Accumulator.
CY/L	Carry/Link Flag in the Status Register.
data	Signed, 8-bit immediate data field.
disp	Displacement; represents an operand in a nonmemory reference instruction or an address modifier field in a memory reference instruction. It is a signed two's-complement number.
EA	Effective Address as specified by the instruction.
E	Extension Register; provides for temporary storage, variable displacements and separate serial input/output port.
i	Unspecified bit of a register.
IE	Interrupt Enable Flag.
m	Mode bit, used in memory reference instructions. Blank parameter sets m = 0, @ sets m = 1.
OV	Overflow Flag in the Status Register.
PC	Program Counter (Pointer Register 0), during address formation, PC points to the last byte of the instruction being executed.
ptr	Pointer Register (ptr = 0 through 3). The register specified in byte 1 of the instruction.
ptr$_{n:m}$	Pointer register bits; n:m = 7 through 0 or 15 through 8.
SIN	Serial Input pin.
SOUT	Serial Output pin.
SR	8-bit Status Register.
( )	Means "contents of." For example, (EA) is contents of Effective Address.
[ ]	Means optional field in the assembler instruction format.
~	Ones complement of value to right of ~.
→	Means "replaces."
←	Means "is replaced by."
↔	Means "exchange."
@	When used in the operand field of the instruction, sets the mode bit (m) to 1 for auto-incrementing/auto-decrementing indexing.
10$^+$	Modulo 10 addition.
∧	AND operation.
∨	Inclusive-OR operation.
⊻	Exclusive-OR operation.
≥	Greater than or equal to.
=	Equals
≠	Does not equal

ISP-8A/600 single-chip 8-bit n-channel microprocessor (SC/MP-II)

physical dimensions



40-Lead Ceramic Dual-in-Line Package (D)



40-Lead Plastic Dual-in-Line Package (N)

ordering information

The SC/MP device may be ordered through the local National Semiconductor sales representative or by contacting our world or international headquarters listed below. The order numbers are as follows:

For an "N" package – ISP-8A/600N
For a "D" package – ISP-8A/600D

National Semiconductor Corporation
2900 Semiconductor Drive, Santa Clara, California 95051, (408) 737-5000/TWX (910) 339-9240
National Semiconductor GmbH
808 Fuerstenfeldbruck, Industriestrasse 10, West Germany, Tele: (08141) 1371/Telex 05-27649
National Semiconductor (UK) Ltd.
Larkfield Industrial Estate, Greenock, Scotland, Tele: (0475) 33251/Telex 778-632

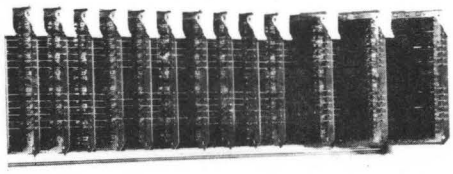
3.8 NIBBLER SUPPORT HARDWARE

3.8.1 72 PIN EDGE CONNECTOR

$5.00

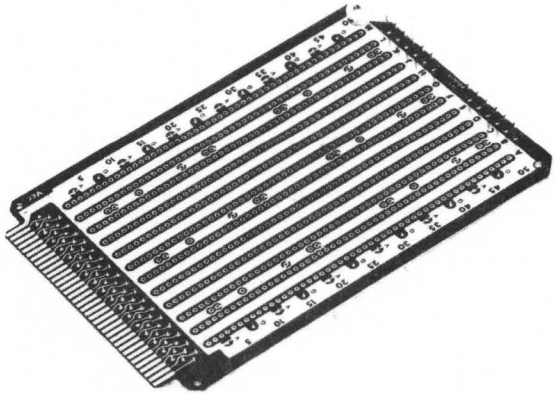
3.8.2 14 POSITION CARD FILE

$39.95



3.8.3 PROTOTYPING BOARD

$39.95



3.8.4 EXTENDER BOARD

$29.95

3.8.5 NIBBLER POWER SUPPLY AND INTERFACE BOARD $39.95

SOON TO BE RELEASED

4. SUPPORT DOCUMENTATION

4.1 INTERFACE AGE ARTICLE ON NIBL. THIS ARTICLE HAS BEEN REPRODUCED WITH THE PERMISSION OF INTERFACE AGE. THE ARTICLE APPEARED IN VOLUME 2 ISSUE 2 JANUARY 1977.

An Inside Look Into NIBL – Extended Tiny BASIC for the SC/MP

By Mark Alexander

National Semiconductor Corp.
Santa Clara, CA

INTRODUCTION

NIBL (National Industrial Basic Language) is a conversational programming language for the SC/MP. It is a language similar to Tiny BASIC Extended, but it also has some unique features. Many of these features, such as a genuinely useful control structure (the PASCAL-influenced DO/UNTIL) and the indirect operator ("@") have been added to the language to allow NIBL to be nearly as flexible as machine language in such applications as medium-speed process control.

By using NIBL, one trades the high execution speed and low memory consumption of machine language for some very tangible advantages: program readability and modifiability which are truly difficult to achieve in machine language programs.

NIBL programs are interpreted by a large (4K byte) TBX SC/MP program that resides in ROM. The interpreter is broken into two main blocks: a program written in an Intermediate (or Interpretive) Language — IL for short — which does the actual interpretation; and a collection of SC/MP machine language subroutines invoked by the IL program. The IL approach is well-documented in vol. 1, no. 1 of ***Dr. Dobb's Journal of Computer Calisthentics and Orthodontia***, and readers should refer to that issue for a more detailed description of the interpretation process.

In Table 1, the formal grammar for NIBL is given. This is the ultimate authority (other than the interpreter itself) on how legal NIBL statements are formed. The following descriptions of the NIBL statements will refer to portions of the grammar. Table 2 contains a list of the error messages produced by the NIBL system. Finally, a listing of the NIBL interpreter is given.

Table 1: NIBL grammar

On reading the 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 repetitions; the question mark, "?", means zero or one occurrences; 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'
                    ! '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
                    ;

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 = (String ! Rel Exp ! '$' Factor) ;

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

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

Table 2: NIBL error messages

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.

AREA	No more room left for program in current page
CHAR	Character after logical end of statement
DIVØ	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

HISTORY OF NIBL

NIBL came into this world as an interpreter for Tiny BASIC, as originally described in the first issue of Dr. Dobb's Journal. That program was written by Steve Leininger, who subsequently left before the program was ever assembled or executed. The current version of NIBL is an almost complete re-write of the original interpreter, with changes and additions being made to improve the modularity of the program, to greatly increase execution speed, and to extend the capabilities of the language itself.

SYSTEM REQUIREMENTS

The NIBL interpreter is intended to be a ROM-resident program in the first 4K of the SC/MP Memory address space (although it will run just as well in RAM). The interpreter requires at least 2K bytes of RAM starting at address 1000 (base 16), of which the interpreter uses nearly 300 bytes for stacks, variables, etc., leaving the rest for the user's program. Another 2K bytes of memory may be added to fill up this 4K page, forming what is hereafter referred to as "Page 1."

The SC/MP architecture forces memory to be split into pages of 4K bytes each; therefore, NIBL allows seven such pages to be used for storing programs. NIBL programs in the seven pages are edited separately, but may be linked together during program execution by special NIBL statements described in the following pages. Page #1 as mentioned above, must be RAM since the interpreter uses part of it as temporary storage; the part used to store application programs starts at location 111E (base 16).

The other six pages of memory, each of which starts at location n000 (base 16), where n is the page number, may be either RAM or ROM. Page 2 is a special page: it can contain a NIBL program to be executed immediately upon powering up the NIBL system.

The memory organization of NIBL is shown in Figure 1.

Throughout this article, the assumption is made that the user has a teletype with paper tape reader and punch, as with the SC/MP Low Cost Development System (LCDS). In fact, NIBL was designed to use the SC/MP LCDS teletype interface, but to be completely independent of the LCDS firmware. If NIBL is to be run on its own, the system should have the same configuration for the teletype, with the reader relay being operated directly by the SC/MP. At present, paper tape is the only medium for saving NIBL programs, but as soon as the hardware and software for a SC/MP cassette interface become available, NIBL will be able to link to routines for saving and loading programs with ease.

Since the teletype interface is not based on a UART, the terminal baud rate can only be changed by modifying the timed delays in NIBL's I/O routines. NIBL has been run successfully at 1200 baud with a CRT terminal; the assembly listing of the NIBL program is programmed for a 110 baud rate system.

COMMUNICATING WITH NIBL

When the NIBL system is ready to accept input, it prompts at the teletype with a ">" prompt sign. (NIBL is now in "edit mode.") The user then enters a line terminated by a carriage return. There are several special characters that are used to edit lines as they are typed:

Shift/O (back-arrow) causes the last character typed to be deleted.

Control/H (backspace) performs the same function as shift/O, but echoes as a backspace/space/backspace sequence, which is only useful if the teletype routines are modified to run at high baud rates for use with CRT's.

Control/U (echoed as " ' U") causes the entire line to be deleted; NIBL reprompts for a new line.

Entering a line to NIBL without a leading line number causes the line to be executed directly by NIBL. Most NIBL statements, as well as the four program control commands, may be executed in this manner.

A line with a leading number (in the range 0 through 32767) is entered into the NIBL program in the cur-

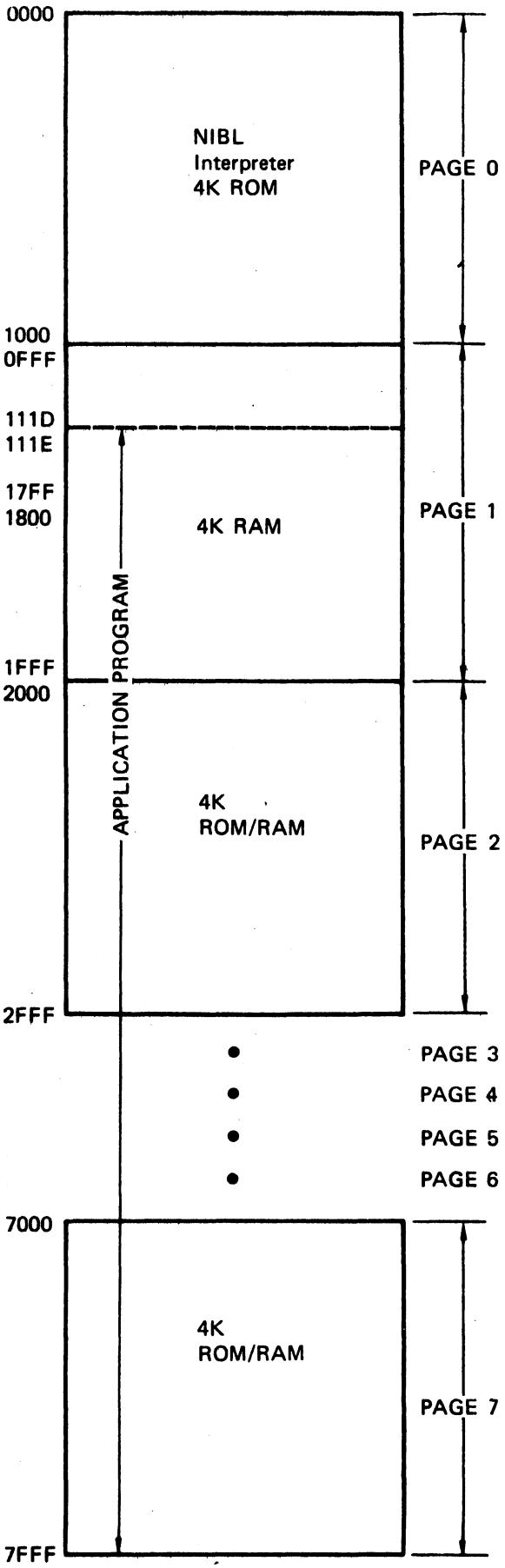


FIGURE 1. NIBL MEMORY ORGANIZATION

rent page. (Make sure that the value of the pseudo-variable PAGE is valid, so that the line isn't lost into nonexistent memory.) The NIBL editor sorts the program lines as they are entered into ascending order by line number.

Typing a line number followed by a carriage return deletes that line from the program. Typing a line with the same number as an existing line's causes the new line to replace the old one in the program.

Each of the seven memory pages may contain a different program, separate from the rest. Editing the program in one page will not affect the other pages. To switch editing from one page to another, simply type PAGE = n, where n is the number of the new page.

VARIABLES

There are twenty-six variable names in NIBL: the letters A through Z. They are all 16-bit binary variables, so they can be used to hold addresses as well as signed numeric data. Space for pre-declared variables is allocated for them in RAM when NIBL powers up.

CONSTANTS

NIBL allows either decimal or hexadecimal (base 16) constants to appear in expressions. Decimal constants must lie in the range 0 through 32767; the unary minus ("−") is used to obtain negative values. The value −32768 is a valid NIBL integer, but it is not legal as it stands. To represent it, use −32767-1 or #8000 instead.

Hexadecimal constants are denoted by a pound sign ("#") followed by a string of hexadecimal digits (0-9, A-F). NIBL does not check for overrun in hex constants; consequently, only the 4 least significant digits of the hex digit string are kept.

FUNCTIONS

NIBL includes three functions that may appear in any expression. These are described as follows:

RND (X, Y) returns a pseudo-random integer in the range X through Y, inclusive, where X and Y are arbitrary expressions. In order for the function to work properly, the value of Y − X should be positive and no greater than 32767.

MOD (X, Y) returns the absolute value of the remainder from X divided by Y (where X and Y are expressions).

TOP (with no arguments) returns the address of the first free byte in the memory page currently being edited or executed. In other words, it is the address of the last byte at the top of the current NIBL application program in the current page, plus one.

PSEUDO-VARIABLES

NIBL has two pseudo-variables in addition to the standard variables. These are STAT and PAGE. Both of these variables may appear on either side of an assignment statement.

STAT represents the SC/MP status register. The current value of the status register can be referred to by using STAT in an expression; or an assignment may

be made to the status register by executing a statement such as STAT = 4 or STAT = STAT OR #20. When NIBL makes an assignment to the status register in this manner, it clears the interrupt-enable bit of the value before it is actually assigned. Note also that only the lower byte of the value is assigned, the high byte is ignored.

The carry and overflow bits in STAT are meaningless since the NIBL system is continually modifying them. The utility of STAT lies in the fact that 5 of its bits are connected to I/O sense lines on the SC/MP chip.

The pseudo-variable PAGE contains the number of the memory page currently being executed or edited. As indicated in Figure 1, there are seven pages in which NIBL programs may be stored; therefore, the PAGE number may be only in the range of 1 through 7. If an assignment of a value outside this range is made to PAGE, only the 3 least significant bits LSBs of the value are used. Page one is defined as LSBs = 000, page two as LSBs = 001 etc.

If PAGE is modified while NIBL is in edit mode, all subsequent editing will take place in the new page.

If PAGE is modified by a NIBL program during execution, control will be passed to the first line of the NIBL program in the new page. This transfer would be effected by a statement such as PAGE = 6 or PAGE = PAGE + 1. Thus, several NIBL programs residing in different 4K pages may be linked together as one large program, if need be. This would allow one to write a 28K STAR TREK program in NIBL, a Herculean and indeed foolish task.

Control may also be transferred from one page to another by three other statements: RETURN, NEXT, and UNTIL. Thus, the first part of a subroutine or loop may be in one page, and the second part may be in another (with control being transferred between the two parts by an assignment to PAGE). In these three special cases, NIBL automatically updates the value of PAGE as the statements are executed.

RELATIONAL OPERATORS

NIBL provides the standard BASIC relational operators, for comparing the values of integer expressions. The operators are as follows:

= equal to
<= less than or equal to
>= greater than or equal to
<> not equal to
< less than
> greater than

All of these operators produce 1 as a result if the relation is true, and 0 if the relation is false. Note that the relational operators may appear anywhere that an expression is called for in the NIBL grammar, not only in IF statements.

ARITHMETIC OPERATORS

NIBL provides the four standard arithmetic functions: addition (+), subtraction or unary minus (−), multiplication (*), and division (/). Since only integers are allowed in NIBL, all quotients are truncated (the MOD function can be used to obtain remainders from division). Any overflow or underflow (other than division by zero) is ignored by NIBL; the reasoning behind this is that it may often be necessary to treat NIBL expressions as unsigned values, such as when performing calculations using memory addresses as the operands. Thus the value of 32767 + 1 is −32768 (or in hexadecimal, °7FFF + 1 = #8000, which makes more sense).

LOGICAL OPERATORS

In NIBL, there are three logical operations that may be performed on values: AND, OR, and NOT. The first two are binary operators, and the latter is unary. All three perform bitwise logical operations on 16-bit arguments, producing 16-bit results. AND, OR, and NOT are sufficient to simulate any other logical operation, through various combinations of the operators.

THE INDIRECT OPERATOR

The indirect operator "@" realizes the functions of PEEK and POKE operations in other BASICs, but with somewhat more elegance. The "@" sign followed by an address (which can be a constant, variable, or expression in parentheses) denotes the contents of that address in memory. Thus, if memory location 245 (decimal) contains 60, the statement X=@245 would result in the value 60 being assiged to X. The indirect operator may also appear on the left side of an assignment statement. For example, @X=@(Y+10) would result in the memory location pointed to by X being assigned the value of the memory location pointed to by the value Y+10.

Use of the indirect operator is not limited to reading from or writing to memory: it also provides a simple way to communicate with peripheral devices that are interfaced to the SC/MP through memory addresses. Note that the "@" operator can only access memory one byte at a time, and that when an assignment is made to a memory location, only the low order byte of the value is moved to the location; the high order byte is ignored.

The indirect operator can also be used to simulate arrays in NIBL. For example, if we wish to define an M x N matrix of one-byte positive integers, we can access the (I,J)th element of the matrix (assuming that (0, 0) is a legal element in the matrix) with the expression @(A+I*N+J). An assignment could be made to that same element by placing the expression on the left side of an assignment statement.

EXPRESSIONS

Expressions in NIBL are composed of variables, constants, function references, pseudo-variables, and operators. Operators bind these other elements together to form complete expressions. NIBL expressions are all 16-bit integers. Evaluation of expressions takes place left-to-right, and the order in which operations take place is determined by operator precedence and the presence of parentheses. The order of evaluation can be deduced from the grammar in Table 1; here is a table of operator precedence:

Lowest precedence (applied last): <, >, <=, >=, =, <>
+, −, OR
*, /, AND

Highest precedence (applied first): @, NOT

Despite this, it is still safest to use plenty of parentheses in expressions to make the intent clear.

PROGRAM CONTROL COMMANDS

LIST causes the entire program in the current page to be listed. Listing can be halted by hitting any key on the teletype: the BREAK key works best.

LIST <number> causes listing to begin at the given line number (or the nearest one greater than the number), rather than at the first line.

LISTing a program is the method used to save it on paper tape. To accomplish this, type LIST with the punch off, then turn on the punch and hit carriage return. After the program is dumped, type a Shift/O with teletype on LOCAL so that the last character (a ">") will be deleted when the tape is entered to NIBL at a later time. NIBL will accept a tape made in this fashion at any time during edit mode. The tape reader is enabled at all times by NIBL, and it does not distinguish between the reader and the keyboard when accepting input. Superfluous line-feed and null characters on the tape are echoed but ignored.

RUN causes three actions: first, all variables are zeroed; secondly, all stacks (the FOR, DO, and GOSUB stacks) are cleared; and finally the program in the current page is executed, starting with the first line in sequence.

RUN is not the only way to start program execution: GOTO and GOSUB can also be used to jump into a program from edit mode. For example, if there is a subroutine at line 1000 that is being tested, typing GOSUB 1000 will cause that routine to be executed, with NIBL returning to edit mode upon encountering a RETURN statement. When GOTO and GOSUB are used to run a program, the variables and stacks are not cleared.

Hitting any key while a program is being run will cause NIBL to break execution, printing a message and the line number where the break was detected. The BREAK key on the teletype works best for this.

CLEAR causes all variables to be zeroed and the three stacks mentioned above to be cleared. This latter feature of the CLEAR command is quite useful after a stack nesting error has occurred (for example, if GOSUBs are nested more than eight levels deep).

NEW clears the program in Page 1, and changes the value of PAGE to 1. This is the form of the command most likely to be used by NIBL novices who do not wish to be confused by the page selection features of NIBL. NEW should be the first thing one types in to NIBL when first powering up.

NEW <number> sets the value of PAGE to the <number>, and clears the program in that page.

ASSIGNMENT STATEMENTS

Previously, two different types of assignment statements have been described: assignments to the pseudo-variables STAT and PAGE, and assignments to memory locations with the indirect operator. Another form of the assignment statement is the conventional assignment to a variable (A − Z), e.g. A = A + 1 or A = 32 < (4 * I). There are also statements which look like string assignments, but these are not standard BASIC, and are described later in the section on string handling. The word "LET" is optional in front of any assignment statement (leaving it out increases execution speed, unlike most Tiny BASIC systems).

IF/THEN STATEMENT

The IF statement allows conditional execution of one or more statements (as many as can fit on one line). The syntax for the IF statement is:

'IF' Rel-exp 'THEN'? Statement

which indicates that the word THEN is optional, and that any statement (including another IF statement) may follow the conditional expression. If the IF condition is true (i.e. is non-zero), the statement following it (and any others on the line) will be executed; otherwise, control immediately transfers to the next program line. The condition does not need to contain relational operators: a statement such as IF MOD (A, 5) THEN . . . is perfectly legal. In this example, the statement following the THEN would be executed if A were not divisible by 5.

GOTO, GOSUB, AND RETURN STATEMENTS

The syntax for the GOTO statement is 'GOTO' followed by an expression. The effect of the GOTO statement is to transfer control to the line whose number is indicated by the expression. An error occurs if the specified line does not exist in the current page. Unlike standard BASICs, any arbitrary expression can be used to specify the line number, as well as the usual decimal constant. This allows computed branches to be performed with the same effect as the ON . . . GOTO statement in standard BASIC.

The GOSUB statement is identical to the GOTO statement in form. It too causes a branch to a new line, but it also saves the address of the following statement on a stack. When a RETURN statement is executed, the saved address is popped from the stack, and control returns to that point in the program. Since an actual address, not a line number, is saved on the GOSUB stack, GOSUB statements may appear anywhere on a multiple-statement line.

GOSUBs may be nested up to eight levels deep; an error will occur if an attempt is made to exceed this limit. The error condition does not destroy the previous contents of the stack, so a RETURN statement can be executed (even in edit mode) without an error occurring. However, any modification of the NIBL program will clear the GOSUB stack, so that a subsequent RETURN without a GOSUB will cause an error.

DO AND UNTIL STATEMENTS

The DO and UNTIL statements are useful in writing program loops efficiently, without using misleading

GOTO statements. Enclosing a group of zero or more statements between a DO statement and an UNTIL <condition> statement (where <condition> is an arbitrary expression) will cause the statement group to be repeated one or more times until the <condition> becomes true (i.e. non-zero). As an example of the use of the DO and UNTIL statements, we present a program that prints the prime numbers:

```
 10 PRINT 1: PRINT 2
 20 I=3
 30 DO
 40 J=I/2: N=2
 50 DO
 60 N=N+2
 70 UNTIL (MOD (I, N) =0) OR (N>J)
 80 IF N>J PRINT I
 90 I=I+2
100 UNTIL 0
```

DO loops may be nested up to eight levels deep, and NIBL acts in the same manner if an overflow occurs as it does with a GOSUB overflow. NIBL also reports an error if an UNTIL statement occurs without a previous DO. A single DO loop may have more than one UNTIL statement as a terminator. For example, if one wished to exit abnormally out of a DO loop and transfer to some appropriate line, it could be done in the following manner:

```
UNTIL 1: GOTO X
```

where X is the line number.

Neither the DO nor the UNTIL statement may be executed in edit mode.

FOR AND NEXT STATEMENTS

The NIBL FOR statement is virtually identical to that in standard BASICs; consequently, it is not explained in great detail here.

As in most BASICs, both positive and negative STEPs are allowed in the FOR statement, and a STEP of +1 is assumed if the STEP portion of the statement is omitted. A FOR loop is terminated by a NEXT <variable> statement, and the <variable> must be the same as that referred to in the FOR statement at the beginning of the loop.

FOR loops may be nested four levels deep; NIBL reports an error if this limit is exceeded, or if a NEXT statement occurs without a previous FOR statement. As with the DO and UNTIL statements, FOR and NEXT may not be executed in edit mode.

Perhaps the only differences between the NIBL FOR statement and that of more elaborate BASICs (such as DEC's BASIC-PLUS for the PDP-11) are that a FOR loop is always executed at least once, and that when a NEXT statement is executed, the STEP value is added to the variable before the test is made to determine if the loop should be repeated (rather than after the test).

INPUT STATEMENT

There are two types of INPUT statements in NIBL: numeric input and string input. The form of the first type is 'INPUT' followed by a list of one or more variables. When this statement is executed, NIBL prompts at the teletype with a question mark ("?"). The user responds with a list of expressions separated by commas, and terminated by a carriage return. For example, a legal response to the statement INPUT A,B,C would be #3FA,26,4*27. These three expressions would then be assigned to the variables A,B, and C, respectively. An illegal response (too few arguments or improper expressions) will result in a syntax error. Any extra arguments in the response are ignored.

The second type of INPUT statement allows strings to be input. The form of the statement is 'INPUT' '$' <address>, where <address> is a Factor, syntactically (usually a variable, constant, or expression in parentheses). When this statement is executed, NIBL prompts the user as before, at which point the user enters a line terminated by the usual carriage return. NIBL then stores the line in memory in consecutive locations, beginning at the address specified. Thus, INPUTS #6000 would cause the input line to be stored starting at location 6000 (base 16); the carriage return would also be stored at the end of the line.

Strings input in this manner can be tested and manipulated by using the "@" operator or the string handling statements described below. They can also be displayed by a PRINT statement.

Neither of the two INPUT statements may be executed in edit mode.

PRINT STATEMENT

The form of the PRINT statement is 'PRINT' or 'PR' followed by a list of print items separated by commas, and optionally terminated by a semicolon, which suppresses an otherwise automatic carriage return after all items in the list are printed.

A print item consists of one of the following:

- A quoted string, which is printed exactly as it appears (with the quotes removed)
- An expression, which is evaluated and printed in decimal format, with either a leading space or a minus sign ("−"), and one trailing space
- A reference to a string in memory, denoted by '$' <address>, where <address> is a Factor as usual. Successive memory locations, starting at the specified address, are printed as ASCII characters, until a carriage return (which is not printed) is encountered.

There is no zone spacing in the PRINT statement, nor does NIBL perform an automatic carriage return/line feed after printing 72 characters. NIBL is not an output-oriented language; fancy formatting has been sacrificed for more useful control structures and data manipulation features. (A subroutine to print a number and skip to the next print zone is a trivial to write in NIBL — it takes about two lines of code, with the DO/UNTIL and FOR/NEXT.)

STRING HANDLING STATEMENTS

String handling in NIBL is very minimal and low-level. The string handling features of the INPUT and PRINT statements have already been mentioned; NIBL provides two more statements for manipulating strings.

A statement such as $<address> = "THIS IS A STRING" would cause the quoted string to be stored in

memory starting at the specified address (which again is a Factor), with a carriage return being appended to the string.

Another statement allows the programmer to move strings around in memory once they have been created. The form of this statement is '$' <destination> '=' '$' <source>, where both <destination> and <source> are Factors, and are the addresses of strings in memory. This statement causes all the characters in the string pointed to by <source> to be copied one-by-one to the memory pointed to by <destination>, until a carriage return (also copied) is encountered. Overlapping the source and destination addresses can produce disastrous results, such as wiping out the entire contents of the current page. Consequently, a string move can be aborted by hitting the BREAK key on the teletype (but it must be done quickly!).

Note that all strings referred to in these statements, and in the INPUT and PRINT statements, are assumed to lie within a 4K page, and wraparound is a possibility which must be anticipated by the programmer. (Long-time SC/MP programmers will be familiar with this minor problem.)

Using these statements, it should be very easy to develop a set of NIBL subroutines for performing concatenation, comparison, and substring operations on strings.

END STATEMENT

The END statement may appear anywhere in a NIBL program and not necessarily at the end. It causes a message and the current line number to be printed, with NIBL returning to edit mode. The END statement is useful when debugging programs, since it acts as a breakpoint in the program that can be removed easily.

LINK STATEMENT

The LINK statement allows NIBL programs to call SC/MP machine language routines at any address. A statement of the form 'LINK' <address>, where <address> is an arbitrary expression, will cause the NIBL system to call the routine at that address by executing an appropriate XPPC P3 (Exchange contents of the Program Counter with designated Pointer Register P3) instruction. The user's routine should make sure that it returns by executing another XPPC P3, and that the value of P3 (Pointer Register #3) upon entry to the routine is restored before returning. The routine may make use of the fact that P2 is set by NIBL to point to the beginning of the RAM block used to store the variables A through Z, with each variable being stored low byte first, high byte second. Thus, parameters may be passed between NIBL programs and machine language routines through the variables. Both P1 and P2 may be modified by the user's routines; they are automatically restored by the NIBL system upon return. The user should be careful not to modify RAM locations with negative displacements relative to P2, or the locations with displacements greater than 51 relative to P2. These locations are used by the interpreter.

REMARK STATEMENT

A comment can be inserted into a NIBL program by preceding it with the word REM. REM causes the rest of the line to be ignored by NIBL during execution. Remarks are useful in debugging programs or helping other people to understand them, but of course, they take up valuable memory. (Then again, memory is getting cheaper all the time.)

MULTIPLE STATEMENTS ON ONE LINE

A program line may contain more than one statement, if the statements are separated by colons (":"). Using multiple statements on a single line improves the readability of the program by separating it into small blocks, and uses less memory for storing the program.

It is important to note that an IF statement will cause any statements appearing after it on the line to be ignored if the IF condition turns out to be false. This is the feature that allows a group of statements to be executed conditionally.

A multiple-statement line may be entered without a line number, but NIBL will only execute the first statement on the line, ignoring the rest.

POWERING UP

NIBL is capable of executing a program in ROM in Page 2 immediately upon powering up, without the need for the user to give the RUN command at the teletype. When NIBL initializes, it examines Page 2 and makes an educated guess about the possible existence of a legal NIBL program in that page. If NIBL thinks there really is a program there, it starts executing it immediately; thus, if the program halts for some reason, the value of PAGE will be 2. But if NIBL fails to find a legal program in Page 2 initially, it sets the value of PAGE to 1 (the normal case) and prompts at the teletype.

When executing programs, NIBL periodically checks for keyboard interrupt, returning to edit mode if it detects it. Therefore, if a NIBL program is to be executed with the teletype disconnected, the Sense B line of the SC/MP should be set high so that NIBL will not sense an interrupt while running. This would allow a NIBL system to act as a process controller which starts executing immediately upon powering up.

4.2 BIBLIOGRAPHY OF SUPPORTING DOCUMENTATION

4.2.1 SC/MP INSTRUCTION GUIDE

4.2.2 SC/MP TECHNICAL DESCRIPTION

4.2.3 SC/MP MICROPROCESSOR APPLICATIONS HANDBOOK

4.2.4 SC/MP MICROPROCESSOR ASSEMBLY LANGUAGE

THESE PUBLICATIONS ARE AVAILABLE UPON ORDER FROM DIGI-KEY. THEY WILL PROVIDE ADDITIONAL INFORMATION ON THE SC/MP AND PROGRAMMING OF THE SC/MP.

5. TESTING YOUR NIBBLER

IF FOR SOME REASON YOU ARE HAVING DIFFICULTY GETTING YOUR NIBBLER TO WORK, HERE IS A SIMPLE RELIABLE TEST TO SEE IF IT IS OPERATING.

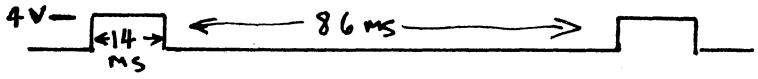
1. APPLY 5 VOLT POWER TO THE NIBBLER AS FOLLOWS. CONNECT +5 VOLTS TO PIN 1. CONNECT THE GROUND (NEGATIVE) LEAD OF THE POWER SUPPLY TO PIN 72.

2. CONNECT PIN 10 TO PIN 72 (I.E. GROUND PIN 10).

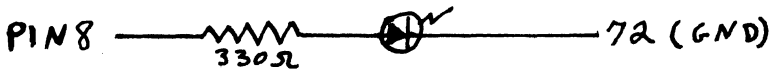
3. DO NOT MAKE CONNECTIONS TO ANY OTHER PINS.

4. MONITOR THE SIGNAL BETWEEN PIN 8 AND GROUND (PIN 72) USING EITHER A SCOPE, OR THE SIMPLE LED TESTER PROVIDED WITH THE NIBBLER.

5. USING A SCOPE YOU SHOULD SEE THE FOLLOWING WAVEFORM.



6. THE LED TESTER CONSISTS OF A MV5053 RED LED AND A 330 OHM 1/4 WATT RESISTOR CONNECTED AS BELOW:



THE FLAT SIDE OF THE LED IS THE CATHODE, AND IS CONNECTED TO GROUND.

THE LED WILL FLASH AT A 10 PULSE/SECOND RATE. THIS IS VERY EASY TO DETECT. IF THE LED GLOWS CONTINUOUSLY OR DOES NOT GLOW AT ALL, THE NIBBLER IS NOT FUNCTIONING. THE IC'S SHOULD BE PRESSED DOWN FIRMLY INTO THEIR SOCKETS AND THE TEST REPEATED. IF THERE IS STILL NO 10 PULSE/SECOND FLASH THE BOARD MUST BE REPAIRED.