Publication Number 420306319-001A
Order No. 420306319-001
November 1980

STARPLEX™

NSC Tiny BASIC User's Manual

REVISION RECORD

REVISION	RELEASE DATE	SUMMARY OF CHANGES
A	11/80	First Release. NSC Tiny BASIC, User's Manual Publication No. 420306319-001

Table of Contents

SECTION I

TABLE OF CONTENTS (Cont'd.)

Page

TABLE OF CONTENTS (Cont'd.)

Section 1

CHAPTER 1

1.1 Bringing Up The INS8073 System

All the examples are based around the example system shown in Section 3, Figure 1-4.

For those of you who have designed your own system, with help from Section 3 of this manual, it is assumed that you have the experience to interpret the following instructions to suit your own system. The sequence below tells how to hook the standard NSC Tiny BASIC card shown in Section 3, Figure 1-4, to a power supply and TTY or CRT to get it running.

Things needed:

Power Supply: +5,
-12V (For serial communications)
Optionally +25 (For PROM programmer)

A power supply cable can be connected directly to the board at the P1 mounting, or stake pins can be inserted and the power supply can be connected through a suitable connector, i.e., MOLEX MXI-9 6471. A cable or connector is attached to Power Supply in the following order:

CONDUCTOR PIN#	1	2	3	4
VOLTAGE	+5	-12	+25	GND

If you're using a TTY, it must be connected for 20mA current loop, as described in its own manual, and will connect to the edge connector fingers in the following manner:

PIN#	1	2	3	4	5	6
SIGNAL	XNT+	XNT-	RCV+	RCV-	RD RLY+	RD RLY-

Pin numbers are etched onto the board, remember that Pin 6 is the one closest to the edge of the card on the side with the componenets.

If you're using a CRT terminal, you should hook it up with a standard MALE-MALE cable, (National's 601305491-001 will do just fine), to the RS-232 (D-type) connector on the board. Make sure your terminal is set to RS-232 (if that's a switch selectable option, if not, just assume it is an RS-232 terminal); and make certain upper case and full duplex are selected.

1.2 Baud Rates

Generally, the higher the Baud Rate, the better, as it means less waiting time for you; however, if you are using a TTY you have no choice. The Baud Rate must be set to 110.

The way you set the Baud Rate is with the two jumpers E18-E19, E16-E17. We can call E18-E19 D0, and E16-E17 D1. Set the Baud Rate on a terminal to the highest rate, or 4800, which- ever is lower, set the jumpers to match it as shown in the diagram below. A "1" signifies that the jumper is missing, a "0" means that it is installed.

	E16-E17 D2	E18-E19 D1
110	0	0
300	0	1
1200	1	0
4800	1	1

After you have done all of this, and double checked it, connect the board to the CRT terminal or TTY, warm up the terminal, hook up the power supply, then turn on the power.

If all went well, you should get a right pointing caret (>) prompt. Push the RESET button and the prompt (>) should appear again. You are now ready to begin using your 8073 system.

CHAPTER 2

2.1 Introduction

The INS8073 is a single-chip computer that directly executes NSC Tiny BASIC, a high-level language. Writing programs in NSC Tiny BASIC offers the following advantages over writing programs in assembly language:

- Programs written in NSC Tiny BASIC eliminate the need for memory consuming Editor, Assembler Monitor/Debug programs. All of these functions are built in.

- Programs may be written and debugged using a small, inexpensive system. Purchase of an expensive development system is not required.

- Program debugging is fast and simple. Program execution may be suspended, variables and other parameters examined/altered, errors corrected, and execution resumed at the point where it was suspended - all without the need to reassemble or reload the program. (NSC Tiny BASIC programs do not have to be assembled.)

- Programs can be written in one tenth the time of equivalent assembly language programs due to the power of the NSC Tiny BASIC language, its English-like simplicity and built in edit/debug capability. Programs are also easy to maintain because they are self documenting.

- Programs are relocatable: they may be loaded and executed anywhere in memory without modification.

- Program memory can be quickly checked for valid code because NSC Tiny BASIC programs are stored as a sequence of ASCII characters. (Executable assembly language programs are considerably more difficult to check because they are stored in memory as a sequence of binary numbers).

NSC Tiny BASIC was designed for use on the INS8073 single-chip micro-interpreter, a product of National Semiconductor Corporation. NSC Tiny BASIC is a simplified version of the computer language, BASIC, "Beginners All-purpose Symbolic Instruction Code", developed by Dr. John Kemeny and Dr. Thomas Kurtz at Dartmouth College in 1963. BASIC has become the "People's Computer Language" 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, hobbyists, teachers, college students, and a vast multitude of young people in elementary and secondary schools.

The original NSC Tiny BASIC was designed for applications such as integer arithmetic problems, computer games and teaching beginners how to program computers. NSC Tiny BASIC has extended capabilities that make it a powerful design tool for developing control applications.

Information on NIBL upon which NSC Tiny BASIC was first published in People's Computer Company, Volume 3, Number 4 (March 1975) and Volume 4, Number 1 (July 1975). The best source of information on Tiny BASIC is Dr. Dobb's Journal of Computer Calisthenics and Orthodontia, beginning with Volume 1, Number 1 (January 1976) and continuing through several issues.

This book is designed to help you teach yourself how to use NSC Tiny BASIC and the INS8073: it consists of three major sections:

SECTION 1: A primer designed for self study. This self teaching primer presents the elements of NSC Tiny BASIC in a step-by-step manner. It is assumed that the reader has access to an INS8073-based system and will try out the examples and exercises as they are presented in the primer. It is also assumed that the reader has no previous computer programming training or experience, but is experienced in electronic hardware design using non-computerized techniques.

SECTION 2: A guide that provides quick reference to information for people who have worked through the primer, or, who already know how to program in some form of BASIC.

SECTION 3: A description of a typical INS8073 system: details on setting up the computer system and getting NSC Tiny BASIC running. Section 3 assumes that the reader has a prior knowledge of digital electronics: and, this section gives schematics and a description of an example 8073 NIBL-II demonstrator card.

CHAPTER 3

3.1 Introduction

The INS8073 is a "task-oriented" microinterpreter. NSC Tiny BASIC is the language that instructs the system to perform various and sundry functions.

The use of microcomputers to control electronic, electrical and electromechanical devices is very much an engineer's dream come true. A computer works from a written out specification of what the completed device is supposed to do. This specification, written in a very exact and unambiguous style, is called a program. As with specifications and schematics there are conventions about exactly how a program is to appear. This set of conventions is called a language. The language used on this computer is a version of BASIC called NSC Tiny BASIC.

When setting out a schematic for someone who is not up to your background in electronics, you have to spell everything out in more detail than you would for a colleague who is right with you. Until a computer knows as much as you want it to know, everything must be spelled out in a meticulous and precise manner. Once these instructions are spelled out - that's it; the computer will henceforth do it right every time.

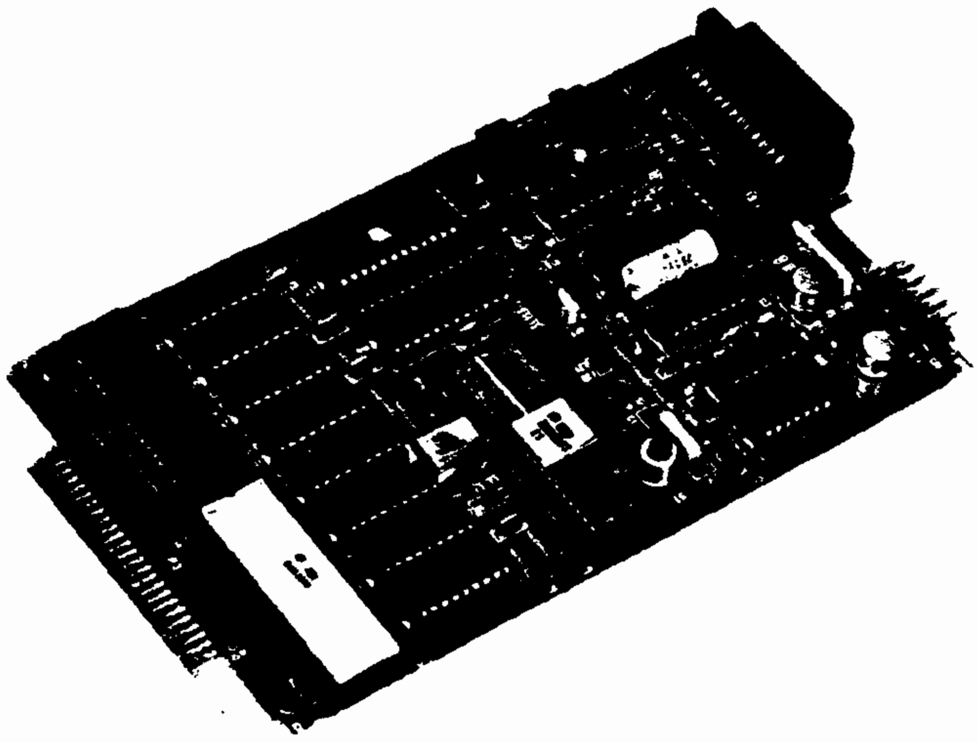


Figure 3.1. INS8073 Based System

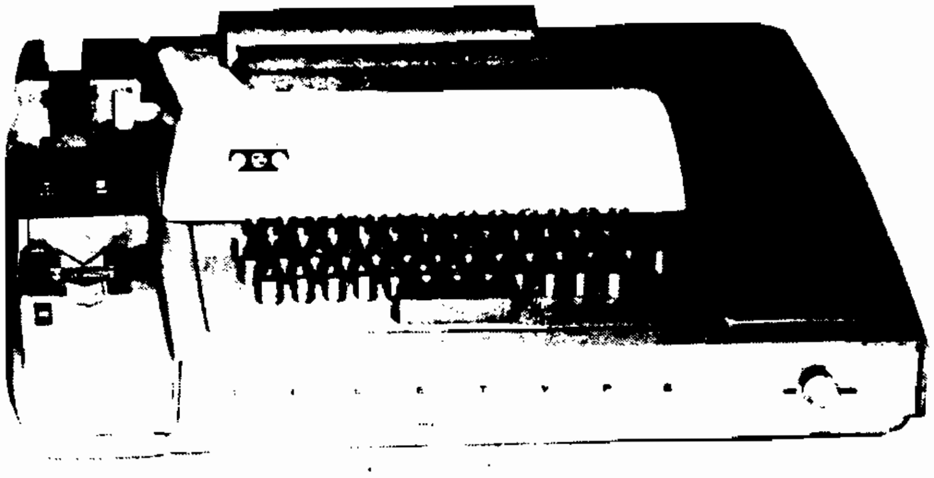
3.2 TTY/CRT Terminal

You will probably be using a Model 33 Teletype or a CRT (Cathode-Ray-Tube) terminal to communicate with your INS8073. In the following text, TTY (Teletype) and CRT (Cathode-Ray-Tube) are used interchangeably.

The letters of the Roman alphabet and Arabic numerals were invented long before computers when nobody cared that the letter "O" looked just like a zero. It is, however, very important for the computer to tell them apart; therefore, the numeral zero is written as an "O" with a slash through it (Ø). The letter "O" is left alone. Most Teletypes will print the zero character with a slash and an "O" without a slash; check your teletype to make sure it observes this convention.

When programming, sometimes you will type to the computer, sometimes the computer will type to you. When it is the computer's turn, it just goes ahead and types. When the computer is "thinking" it acts as if you were not there. When it is your turn to type the computer prompts you by typing the character ">" on the left margin of the paper/screen. The right pointing caret (>) is called the "prompt" character. After typing the prompt, the computer will wait patiently until you type something.

NSC Tiny BASIC recognizes only CAPITAL LETTERS; lower case letters are not used at all. (The Model 33 Teletype doesn't have any lower case letters.) Your CRT may or may not have lower case; if it does, switch the upper/lowercase switch to upper case.



Fiqure 3.3 A Typical TTY

Fiqure 3.2. A Typical CRT

3.3 Beginning Instructions

Think of something you know how to do like bicycling, skiing, playing piano or designing circuitry. One thing is certain: there are no books in the world that can teach someone how to do any of these things. Books can help, but without getting on a bike, putting on skis, practicing scales, or designing hundreds of circuits and trying them out, a novice can't do any of these things. Same way with programming.

The only way to learn programming is by doing it. With bicycling or skiing you may end up with skinned shins; with programming you may experience a dented ego. People don't like to be told they're wrong; unfortunately for the novice programmer, error messages are what he/she will get most frequently from the computer.

For your reference, the NSC Tiny BASIC ERROR CODE SUMMARY is listed below; what it means is that if NSC Tiny BASIC encounters an error condition in RUN command mode, it will print out ERROR followed by an error number. Error numbers are:

Table 3-1. NSC Tiny BASIC Error Code Summary

ERROR NBR.	EXPLANATION
1	Out of memory
2	Statement used improperly
3	Unexpected character (after legal statement)
4	Syntax error
5	Value (format) error
6	Ending quote missing from string
7	GO target line does not exist
8	RETURN without previous GOSUB
9	Expression or FOR-NEXT or DO-UNTIL nested too deeply
10	NEXT without previous matching FOR
11	UNTIL without previous DO
12	Division by zero

3.4 Start Up

Before you power-up, be certain that your system is properly connected and that the Baud Rate Selector is set. Once you have turned on your INS8073 system, the TTY or CRT will type a prompt character (>) to indicate that it is ready to begin. When you are ready to enter a program with line numbers, type the following:

```
        >NEW #address   (hexedecimal address location)
         NEW
```

The above command (NEW #address, NEW) is used:

1. To prepare the computer for a new program with line numbers.

2. For initial power-up.

3. If you RESET your system in the middle of a programming session you may have to use this command. Try to avoid this because you can easily lose all programs in your system's memory.

4. If you wish to store several programs in memory. Each program will have a different hexadecimal address location, for example:

```
Program 1  -  NEW #1000
              NEW

Program 2  -  NEW #4850
              NEW
```

The NEW (carriage return) command erases an old program: the LIST command lists your program and the RUN command runs your program.

Important: when you are finished typing/talking to the computer, you signal by pressing the RETURN key. This indicates that you are finished with your turn.

Type your name and then press the RETURN key: the following is what should happen:

ERROR 4	The computer responds with ERROR 4. ERROR 4 is listed in this chapter and in Appendix C under the Error Code Summary and is a "Syntax Error". This is because NSC Tiny BASIC does not recognize your name as a command.
>	NSC Tiny BASIC then types a prompt (>) to let you know it is still listening and that it is still your turn to communicate.

This is the first example of an error message. It is the one you will see most often, and it means only that you have typed something that NSC Tiny BASIC doesn't understand. NSC Tiny BASIC does not understand your name simply because it is not in its repertory of commands. Examine the following legal commands.

3.5 The Print Instruction

The computer gets jobs done by following instructions. If an instruction is correctly typed, the computer will execute it immediately. (When a computer follows an instruction it is said to obey, or execute that instruction.) One of the most useful instructions is the one that tells the computer to PRINT a desired result or message.

In English we say that antelopes have four legs, but we say that "antelopes" has nine letters. One of the things we do by putting words into quotes is to indicate that we are referring to the words themselves and not their meanings. The computer uses quotes the same way.

For example, suppose, in a boiler installation that the computer is monitoring the water level. If the level begins to get low (but not low enough to warrant automatic shut down) you might want the computer to print: "Warning, the water level is low.". The instruction you desire to give the computer is:

```
>PRINT "WARNING, THE WATER LEVEL IS LOW"
```

Don't forget to press the RETURN button to make the computer execute the instruction.

```
You type:  PRINT "WARNING, THE WATER LEVEL IS LOW"

The microinterpreter types:  WARNING, THE WATER LEVEL IS LOW
```

NSC Tiny BASIC typed what you told it to type: note that the message was enclosed in quotation marks, but they were not printed.

Suppose that the operator in the boiler installation was away from the terminal, or taking a nap, or having a coffee break. In any of these instances he may not see the warning message. The TTY has a bell which may be used as an alarm. (Other terminals may have different audible alarms - a click, beep, buzz etc.) To sound the bell, hold down the key marked CTRL, CNTRL or CONTROL and, while holding it down, press the G key. On most TTYs, the G key has the word BELL on it as a reminder.

```
*****************************************
*                                       *
* To ring the bell, hold CTRL down and  *
*               press G                 *
*                                       *
*****************************************
```

Hold the CTRL key down and press the G key several times: this will allow you to ring the bell several times. You will note that the bells are heard yet nothing is printed on the TTY. (Appendix   shows other non-printing characters which may be useful.)

Bells (CONTROL/G) can be included in a PRINT instruction. Let's use the example of the boiler installation again and print the same warning message, only this time add the bell to be certain that the operator knows there's an important message:

You type:

PRINT "WARNING, WATER LEVEL IS LOW (CTRL GGGGGG)"

Don't forget to press the RETURN key so that NSC Tiny BASIC knows you are through with your instruction.

NSC Tiny BASIC types:

WARNING, WATER LEVEL IS LOW and then rings the bell six times.

3.6 Using The Computer As A Calculator

NSC Tiny BASIC can do integer arithmetic. Try the following examples on your INS8073. Remember to press the RETURN to finish a line of typing.

ADDITION

```
You type:     PRINT 2+3          Use "+" to add.
NSC Tiny BASIC types:  5
```

SUBTRACTION

```
You type:     PRINT 7-4          Use "-" to subtract.
NSC Tiny BASIC types:  3
```

MULTIPLICATION

```
You type:     PRINT 4*7          Use "*" to multiply.
NSC Tiny BASIC types:  28
```

DIVISION

```
You type:     PRINT 48/6         Use "/" to divide.
NSC Tiny BASIC types:  8
```

If you made no typing errors, the above four examples should actually appear on your TTY page as follows:

```
    >PRINT 2+3          The prompts (>) were typed by NSC Tiny BASIC
     5

    >PRINT 7-4
     3

    >PRINT 4*7
     28

    >PRINT 48/6
     8

    >
```

Now try the following divisions.

```
    >PRINT 23/4
     5

    >PRINT 3/2
     1

    >PRINT 4/5
     0
```

Is NSC Tiny BASIC giving wrong answers? No. It is simply doing integer arithmetic. In division, NSC Tiny BASIC produces the integer part of the quotient.

Using the first example above, >PRINT 23/4, this is what happens:

```
                     5    Quotient. This is what you get when you
                 4 / 23   tell NSC Tiny BASIC: PRINT 23/4
                     20
                      3   Remainder. You will be instructed later
                          on in this manual how to compute the
                          remainder.
```

Most industrial control applications, as well as tasks such as word processing and even the programs that make this language work, need only integers. A valve in a refinery may need to be set to one of a hundred positions (many applications only require resolution of two positions - opened and closed). These hundred positions can be represented by the integers 0 to 100 with 0 being closed, 50 being half opened, and 100 allowing full flow.

In NSC Tiny BASIC, integers can range between the limits of -32768 and +32767, inclusive. This allows any measurement or control to be accurate to one part in over 65,000. Few electrical or mechanical devices in control systems require more accuracy. Yet, by appropriate programming, greater accuracy can be obtained if it is necessary.

A good way to learn more about how NSC Tiny BASIC does arithmetic is to use it as an integer desk calculator. As with any desk calculator, it is possible to overflow if you calculate a number too large or small.

NSC Tiny BASIC handles the problem in two ways:

1. If you try to type, not calculate but type, a number greater than 32767 or less than -32767, NSC Tiny BASIC will print an error message. For example:

```
>PRINT 32768
 ERROR 5            Error 5 = Value (format) error

>PRINT -32768
 ERROR 5            Error 5 = Value (format) error
```

2. If you calculate a number outside of this range, no error message will be generated: the numbers just "wrap around". This method of handling overflow is handy on some occasions, but distressing at other times. For example:

```
>PRINT 32766+1
 32767                  This is the expected answer

>PRINT 32767+1
 -32768                 This is NOT the expected answer

>PRINT -32767-1
 -32768                 This is the expected answer

>PRINT -32767-2
 32767                  This is NOT the expected answer

>PRINT -32768-1
 ERROR 5                Remember, you can't type -32768
```

Think of NSC Tiny BASIC numbers being arranged in a circle:

```
                         -1  0  1
                     -2              2
                  -3                    3
                -4                        4
              -5                            5
               .                            .
               .                            .
               .                            .
          -32764                            32764
              -32765                    32765
                -32766                32766
                    -32767        32767
                          -32768
```

From the circle you can see that 32765+7 = -32764. (Moving in a clockwise direction start at 32765 and count off seven places: you should end up at -32764.) Try it on your system.

```
        >PRINT 32765+7
         -32764                      Correct
```

To subtract, move in a counter-clockwise direction. For example, -32766-5 = 32765. Again, verify this on your system.

```
        >PRINT -32766-5
         32765
```

NOTE: NSC Tiny BASIC didn't print the "correct" answer (-32771) because -32771 is less than -32768. Calculated values will be correct only if the correct value is in the range of -32768 to 32767, inclusive.

Up to this point you have been shown simple problems with one operation. The following examples are a bit more complicated. The formal rules for how expressions are evaluated are in this chapter in section 3.7: you will understand them better if you experiment on these examples first.

```
        >PRINT 2*3+4
         10

        >PRINT 2*3-4
         2

        >PRINT 2*3+4*5
         26

        >PRINT 2*3-4*5
         -14

        >PRINT 2*3*4*5*6
         720
        >PRINT 2*3*4*5*6*7
         5040

        >PRINT 2*3*4*5*6*7*8
         -25216------------------The correct answer is 40320, too big
                                 for NSC Tiny BASIC.  NSC Tiny BASIC
                                 does not tell you that an incorrect
                                 answer has occurred.
```

If you use only +, - and *, NSC Tiny BASIC will give correct results unless the true result is less than -32768 or greater than 32767.

Try some division problems:

```
        >PRINT 720/2/3/4/5/6
         1                       Correct.  720/2 = 360, 360/3 = 120,
                                 120/4 = 30, 30/5 = 6, and 6/6 = 1.

        >PRINT 1/2+1/3+1/4
         0                       The integer quotients are all zero.
                                 0+0+0 = 0.

        >PRINT 2/3*1000
         0                       Incorrect.  Two thirds of 1000 does
                                 not give zero.  Try it a different
                                 way.

        >PRINT 1000*2/3
         666                     Correct.
```

3.7 The Use of Parentheses

The following examples illustrate the use of parentheses in numerical expressions. Verify them on your INS8073.

```
        >PRINT 2*(3+4)
         14

        >PRINT (2+3)*(4+5)
         45

        >PRINT (2*3+3)*8+7
         79

        >PRINT (47-23)/6
         4

        >PRINT (2+3)/(4+5)
         0
```

NSC Tiny BASIC does not tell you that a computed answer is incorrect because the true result is outside the range, -32768 to 32767. For example:

```
>PRINT 1000*(39-72)
 -32536                         The correct answer is 33000
```

An incorrect result can occur even if the true result is in NSC Tiny BASIC's range. This will happen if an intermediate calculation lies outside the range -32768 to 32767. For example:

```
>PRINT 201*200/2
 -12668                        The correct answer is 20100.
```

In the above example you got an incorrect result because NSC Tiny BASIC first computed 201*200 which has a true result of 40200 and this is outside its range. NSC Tiny BASIC obtained -25336 for this result, then divided by 2.

```
>PRINT 201*(200/2)
 20100                        Correct.
```

Parentheses were used to cause NSC Tiny BASIC to first compute 200/2, then to multiply by 201.

3.7 Rules For Evaluating Expressions

Division by zero (0) stops everything and gives the message:

```
ERROR 12
```

Expressions are evaluated (in the absence of parentheses) by doing all multiplications and divisions from left to right. After they are completed all additions and subtractions are done, again, from left to right. Any fractional results from a division are simply ignored (truncated). The results are not rounded. For example:

```
2/3*1000
```

is evaluated to zero, since the integer part of 2/3 is zero, and zero times 1000 is zero. But:

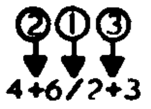
```
1000*2/3
```

evaluates to 666 because 1000*2 is 2000 and 2000/3 is 666.66666, (the fractional sixes to the right of the decimal point are dropped).

The expression 4+6/2+3 evaluates to 10 because the division is done first yielding 4+3+3, and then the additions are done from left to right. In other words, 4+6/3+3 is evaluated:

```
4+6/2+3 = 4+3+3 = 7+3 = 10
```

The order in which operations are done is shown below in still another way. The numbers in the circles show the order:

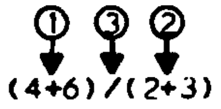


Parentheses override the normal rules. Anything inside a pair of parentheses gets evaluated before that which is outside. This is the normal algebraic convention. Thus:

```
        (4+6)/(2+3)
```

evaluates to 2, thusly: (4+6)/(2+3) = 10/(2+3) = 10/5 = 2

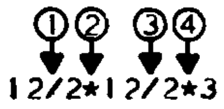
Shown below is the order in which operations are done by the use of numbers in circles.



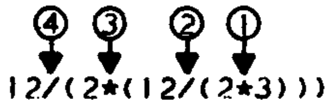
Parentheses may be nested as needed. This means you can have parentheses within parentheses.

```
        12/2*12/2*3 = 6*12/2*3 = 72/2*3 = 36* 3 = 108

        12/(2*(12/(2*3))) = 12/(2*(12/6)) = 12/(2*2) = 12/4 = 3
```

Or, using the circles:



versus



Check these in your head, and then on the computer.

Good programming practice avoids expressions like 12/2*12/2*3 as they are hard to read. It is clearer (and thus less error prone) to write ((12/2*12)/2*3) using spacing and parentheses for clarity even if they are not technically necessary.

Algebraic notation is used in NIBL, modified as necessary to fit on a single line and, of course, to use proper NSC Tiny BASIC arithmetic symbols.

ALGEBRAIC EXPRESSION	NSC TINY BASIC EXPRESSION
$\frac{36}{9 + 3}$	36/(9+3)
$\frac{12 \times 58}{7 \times 25}$	(12*58)/(7*25)
$\frac{120 \times 60}{120 + 60}$	(120*60)/(120+60)

There are limits to the orders of orecedence allowed in any one line. These, however, are hard to exnlain, or even find. The rule of thumb is that if you get an "ERROR 9" occurring after a particularly long expression, try to break that expression into two or more parts.

3.8 Mistakes

Perhaps the deadliest assumption in engineering design is that anybody using the equipment will use it correctly. NSC Tiny BASIC provides error messages after it is too late. If you are working on a TTY and are lucky enough to catch yourself in the middle of a statement, having just typed an incorrect character, you do not have to throw away the good part and retype the whole thing.

The first mistake correcting facility is a sort of backspace. Say that you typed "PRINR" instead of "PRINT". If, after the "R" you held down the SHIFT key and pressed the letter "O" you would get a left pointing arrow or underline. This means that the last letter you typed (the "R") is deleted and you can now type the correct letter ("T"). Try it a few times.

```
        >PRINR 2+3
         ERROR 4            PRINT misspelled

        >PRINR_T 2+3        After typing R, type __(SHIFT O), which
                            erases the R. Then type the rest of the
                            line. Everything is OK to NSC Tiny BASIC,
                            although it looks wrong on your TTY.
```

The backspace feature can be used repeatedly. It is up to you to keep track of just how many letters have been obliterated.

```
        >PRINT 3+2___5*98___5
         25
```

A true backspace feature is provided for use with CRT terminals. Pressing the backspace key (or Control H) will erase the last character from the screen and memory.

If you want to cancel an incorrect line entry without having to wait for the error message, hold the CTRL button and strike the letter "U". NSC Tiny BASIC will type ^U, do a carriage return line feed, then it will type the prompt (>).

```
>TYPE AN INCORRECT LINE ENTRY AND PRESS "RETURN" and get
 ERROR 4

>

>TYPE AN INCORRECT LINE ENTRY AND PRESS CONTROL U^U

>  ←————No syntax error.
```

If you are lucky enough to be using a CRT, just backspace and retype the offending character.

3.9 Exercises

Complete the following:

1. In NSC Tiny BASIC, numbers are integers in the range _________to_________, inclusive.

2. If you type: PRINT "TURN SWITCH NO 3 ON"

   NSC Tiny BASIC will type:_________________________

3. If you type: PRINT 7*7

   NSC Tiny BASIC will type:_________________________

4. If you type: COME ON NSC TINY BASIC. GET WITH IT!

   NSC Tiny BASIC will type:_________________________

Do the following in your head or with paper and pencil, as you think NSC Tiny BASIC would do them. Then, verify your answers.

5. 2*3+4*5+6*7 =__________________________

6. 123*(42/127) =__________________________

7. 1000*1000 =_____________________________

8. 22/7*1000 =_____________________________

9. 1000*22/7 =_____________________________

You will find the answers to these exercise questions in Appendix A.

CHAPTER 4

4.1 Variables

If, instead of typing:

```
        PRINT 120/4/5
```

you typed:

```
        A=120/4/5
```

the result (which is 6, as the expression is evaluated from left to right) would be given the name A, A is called a variable. The instruction:

```
        PRINT A
```

would result in the value 6 being printed. The following is the entire sequence of instructions as they might appear on your TTY page or CRT screen.

```
        >PRINT 120/4/5
         6

        >A=120/4/5

        >PRINT A
         6
```

Try another one.

```
        >A=7            The value 7 was assigned to the variable
                        A and the value 5 to the variable B.
        >B=5            Since A=7 and B=5, A+B will be 12.

        >PRINT A+B
         12
```

NSC Tiny BASIC now is instructed to know A=7 and B=5.

```
        >PRINT A*B
         35             A=7 times B=5  =  35
```

In NSC Tiny BASIC there are 26 variables, the letters of the alphabet A through Z. Each variable may be best considered as a pigeonhole in which exactly one number can be stored. When it is stated that K=4325, it means to replace any prior value that K may have had with the new value 4325. The old value is lost. The instruction G=T tells the computer to make a copy of whatever value is in T and to place that copy in pigeonhole G. In computer jargon the pigeonholes are called "memory locations" because they can "remember" values.

Later you will see that many more locations are available to store data in, but for now there are only 26 variables in NSC Tiny BASIC:

```
        A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Before a variable has been assigned a value (jargon for putting a number into a pigeonhole), NSC Tiny BASIC gives it the value Ø. It is as if just before you sat down to use the computer someone had typed:

```
        A=Ø  B=Ø  C=Ø etc.
```

When you first start NSC Tiny BASIC all the variables will contain the value of zero (Ø).

Skeptical? Try it out on your system.

```
        >PRINT A
         Ø

        >PRINT B
         Ø

        >PRINT C
         Ø
```

and so on, if you wish, up to PRINT Z.

Up to now you have used PRINT statements that print only one thing.

```
        >PRINT 7                  One thing (7).
         7                        One thing (7).

        >PRINT 2+3                One thing (2+3).
         5                        One thing (value of 2+3).

        >A=13

        >PRINT A                  One thing (A).
         13                       One thing (value of A).
```

The PRINT statement can print more than one thing:

```
        >PRINT 7,5                Two things (7 and 5).
         7  5                     Two things (7 and 5).

        >PRINT 7+5,7-5            Two things (7+5 and 7-5).
         12  2                    Two things (values of 7+5 and 7-5).

        >A=7

        >B=5
```

```
>PRINT A,B                  Two things (A and B).
 7  5                       Two things (values of A and B).

>PRINT 7+5,7-5,7*5,7/5      Four things.
 12  2  35  1               Four things.

>
```

```
NOTE:  PRINT 7+7,7-5,7*5,7/5
                 ↖  ↑  ↗
                 COMMAS
```

You can print two or more things provided you separate each thing to be printed with a comma in the PRINT statement.

4.2 Exercises

Pretend for a few minutes that you are the INS8Ø73 and that NSC Tiny BASIC is the language you understand. Show what would happen if your user typed the following:

ONE

```
>A=7

>B=5

>PRINT A+B,A-B,A*B,A/B

 ____ ____ ____ ____
```

TWO

```
>M=47

>N=9

>Q=M/N

>R=M-N*Q

>PRINT M,N,Q,R

 ____ ____ ____ ____
```

THREE

```
>A=2

>B=3

>C=4

>D=5

>PRINT A*B+C*D,(A+B)*(C+D)

 ____ ____
```

FOUR

```
>A=37

>Q=A/1Ø

>R=A-1Ø*Q

>B=1Ø*R+Q

>PRINT A,B

 ____ ____
```

FIVE

```
>R=32

>PRINT R*22/7,(R*R)*22/7

 ____ ____
```

You will find the answers in Appendix A

4.3 The Stored Program

Compute the squares of 23, 37, 53 and 88. That is, compute:

$$23^2,\ 37^2,\ 53^2 \text{ and } 88^2.$$

```
>PRINT 23*23                 23² = 23 * 23
 529

>PRINT 37*37                 37² = 37 * 37
 1369

>PRINT 53*53                 53² = 53 * 53
 2809

>PRINT 88*88                 88² = 88 * 88
 7744

>
```

You can give more of the work to NSC Tiny BASIC: do this by storing a program to compute the square of a number...don't do it yet.

```
10 X=23

20 PRINT X*X
```

If you did type this is and got an ERROR ! message here, it's because your RAM is not at the default location. To remedy this situation, you must tell NSC Tiny BASIC where your RAM is with a NEW statement. If your RAM is at hexadecimal 1000, then you would enter NEW #1000 then NEW again. For example:

```
>NEW #1000
 NEW
```

Notice that the above program consists of two statements and that each statement begins with a line number.

```
10 X=23
↑
Line Number.  A line number can be an integer from
              0 to 32767.
```

When statements with line numbers are typed, the statements are not executed immediately. Instead, the statements are stored in memory for later execution.

Before you store the above program, clear out - or erase - any old program that might be in memory. To do this type:

```
NEW #1000
NEW
```

NOTE: NEW #1000 sets the start of program pointer at location #1000 hexadecimal. The number symbol (#) is important, this will be fully discussed in Chapter 5.

It is important that the start of program pointer is set to the beginning of available RAM. This allows the program lines to be stored as they are typed in. If your 8073 system differs from the one described at length in Section 3: determine the start address of the RAM in your system: then, use that address in your "NEW" command.

NSC Tiny BASIC will erase any old program in its memory and get ready to accept your new program.

```
>NEW

>                           NSC Tiny BASIC is ready for a new
                            program.
```

Store the program to compute the square of a number. Type the following (except for the prompts - NSC Tiny BASIC does that for you.).

```
>NEW
>10   X=23
>20   PRINT X*X
>
```

The program is now stored in memory. To verify this:

Type LIST and press the RETURN key.

```
>LIST◄───────────────── When you type LIST, NSC Tiny BASIC
 10  X=23               lists the program.
 20  PRINT X*X
>
```

To get a copy of the program currently stored in the INS8Ø73's memory, type LIST and press the RETURN key. RUN the program.

```
>RUN
 529
>
```

First NSC Tiny BASIC did this $\longrightarrow$ 1Ø X=23

Then $\longrightarrow$ 2Ø PRINT X*X

That's all, so the INS8Ø73 stopped.

Look over the last few inches of TTY paper: you may find it looks something like the following. (Line spaces have been added to make it easier to read.)

```
>NEW                        First you erased any old program in the
                            system.

>1Ø  X=23                   Then you typed in this two line program.
>2Ø  PRINT X*X

>LIST                       Then you asked NSC Tiny BASIC to type
                            the program out.

 1Ø X=23                    NSC Tiny BASIC obliged.  (Note:  No
 2Ø PRINT X*X               prompts.)

>RUN                        Then you gave the RUN command.

 529                        NSC Tiny BASIC ran the program:  this
                            was the result.

>                           Having done its appointed task, NSC
                            Tiny BASIC typed a prompt...ready for
                            more work.
```

Change the value of X. To do this, type in a new Line 1Ø. This will replace the old Line 1Ø with the new Line 1Ø. After making this change, LIST the modified program. Don't type NEW.

```
>1Ø  X=37

>LIST

1Ø X=37                     This is the new Line 1Ø,

2Ø PRINT X*X                and the old Line 2Ø.

>
```

You can replace any line in the program by typing a new line with the same line number. To delete any line from a program, simply type in that line's number followed by a carriage return. When the program is listed, that line will no longer remain. RUN the modified program.

```
        >RUN
         1369

        >
```

4.4 Exercises

1. Change Line 10 to 10 X=53 then LIST the modified program and RUN it.

2. Change Line 10 to 10 X=88 then LIST the modified program and RUN it.

If you did everything on the previous two pages without making any typing errors, the TTY page will look like the following. (Again, line spaces have been added for readability.)

REMEMBER

```
NEW

>10 X=23
>20 PRINT X*X
```

1. To erase any old program and get NSC Tiny BASIC ready for a new program, type NEW and press RETURN.

```
>LIST
>10 X=23
 20 PRINT X*X
```

2. To get a typed copy of the program currently in the INS8073's memory, type LIST and press RETURN.

```
>RUN
 529
```

3. To tell NSC Tiny BASIC to execute the program in its memory, type RUN and press RETURN.

```
>10 X=37
>LIST

10 X=37
20 PRINT X*X

>RUN
 1369

>10  X=53

>LIST

 10  X=53
 20  PRINT X*X
```

4. To replace any single line of a program in memory, type a statement with the same line number.

```
>RUN
 2809

>10  X=88

>LIST

 10  X=88
 20  PRINT X*X

>RUN
 7744

>
```

4.5 The - GO TO - Statement

If you typed the instruction:

```
    >PRINT "THE BOAT IS SINKING. MAN THE PUMPS!"
```

and pressed the RETURN key, the computer would print:

```
     THE BOAT IS SINKING. MAN THE PUMPS!
```

and then stop. In a situation where a boat was actually sinking, the computer should be more insistent and repeat the message (complete with bells) until somebody pays attention. There is a way to do this. Type in the following program. First type NEW. (Don't RUN the program yet.)

```
    >NEW

    >10  PRINT "THE BOAT IS SINKING. MAN THE PUMPS! (CTRL GGGGGGG)"

    >20  GO TO 10

    >
```

Before you RUN this program - you must know how to stop it. When you type RUN and press the RETURN key, the TTY will begin running the program and ringing bells. To stop a runaway computer, press BREAK (or any other key) until the computer stops.

Type RUN and press RETURN.

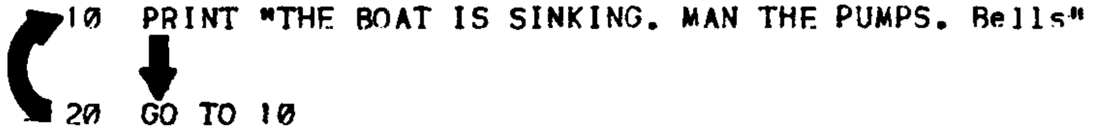
```
    >RUN

     THE BOAT IS SINKING. MAN THE PUMPS! Bells
     THE BOAT IS SINKING. MAN THE PUMPS! Bells
     THE BOAT IS SINKING. MAN THE PUMPS! Bells
     THE BOAT IS SINKING. MAN THE PUMPS! Bells
     THE BOAT IS SINKING. MAN THE PUMPS! Bells
```

To STOP the program, press BREAK.

The following is a short analysis of the above program. Each line has a number. The first line is numbered ten, the second twenty. When you say "RUN" the computer starts to execute lines beginning with the lowest numbered line. In this case that is Line 1Ø: the computer prints "THE BOAT IS SINKING. MAN THE PUMPS! Bells" When it is done with Line 1Ø, it then executes the next higher numbered line. In this case it is Line 2Ø. Line 2Ø has a new instruction, the GO TO instruction, it does the obvious thing and tells the computer what line to go to, i.e., what line to execute next. The computer executes Line 1Ø again, then looks for the next higher numbered line, and so forth. The computer will not stop until it is either turned off, or you stop it by pressing the BREAK button.

If you are still unsure about how the GO TO program works, follow the arrows:

```
>RUN
```



```
1Ø  PRINT "THE BOAT IS SINKING. MAN THE PUMPS. Bells"

2Ø  GO TO 1Ø
```

This program is in the form of a loop. The computer goes around the loop until you press the BREAK key.

After you've stopped the program by pressing the BREAK key, you can start it again by typing:

```
CONT (for continue) then press RETURN
```

The program starts where it left off and continues to print the message over and over again until the BREAK key is again pressed.

The implications of this little program are important: It is a little program, yet it produces a lot of output! Tell a computer to write, "I will do my homework" a thousand times and it will do it uncomplainingly. In an automobile, a microcomputer can be programmed to check the air pressure in the tires, the manifold pressure, fuel flow, battery voltage, the timing and so forth, a hundred times a minute, every minute the car is in operation. Repetitive jobs, however many times they must be done, are usually no more difficult to program than jobs that must be done only once or twice.

4.6 The - INPUT - Statement

Revert back to the problem of computing the value of X^2 for various values of X. The INPUT statement is a handy method for feeding values into variables. Follow along with the program to compute:

X^2, then use it to compute 23^2, 37^2, 53^2, and 88^2.

```
>NEW

>10 INPUT X     (This is the INPUT statement)
>20 PRINT X*X
>30 GO TO 10
```

The above is a three statement program, including a new type of statement called INPUT. RUN the program:

```
>RUN
?               (A new kind of prompt.)
```

NSC Tiny BASIC is now doing the INPUT statement. It types a question mark, then waits. You must type a number and press RETURN.

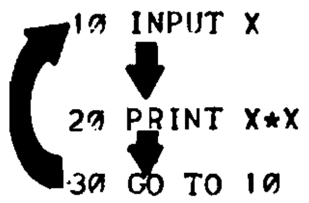
```
>RUN

? 23            (Type 23 and press RETURN.)
 529

?                (NSC Tiny BASIC typed another question mark to
                 show it's ready for more values of X. Continue
                 with 37, then 53, then 88.)

>RUN

? 23
 529

? 37
 1369

? 53
 2809

? 88
 7744

?              NSC Tiny BASIC will keep prompting with ? until
               you let it know that you are finished. To do
               this:
```

Press and hold CTRL and, while holding CTRL down, press C.

```
        ? CTRL/C        NSC Tiny BASIC has stopped running the program
        STOP at 1Ø      and waits for the next command.
        >
```

Remember, NSC Tiny BASIC statements are done in line order number, unless a GO TO breaks that order. In the preceding program, the statements are done in the order shown below. Again, follow the arrows:

```
        >RUN
```



```
        1Ø INPUT X        Program loops around until you stop it
                          by typing CTRL and C together - CTRL/C
        2Ø PRINT X*X

        3Ø GO TO 1Ø
```

The following program computes the value of AX+B for INPUT values of A, X, and B.

```
        >NEW
        >1Ø INPUT A
        >2Ø INPUT B
        >3Ø INPUT X
        >4Ø PRINT A*X+B
        >5Ø PRINT ""        This prints an "empty line".  You could also
        >6Ø GO TO 1Ø        use the expression without the quotes.  They
        >RUN                only serve to make the output prettier.

        ? 2     ---------A
        ? 3     ---------B
        ? 5     ---------X
        ?13     ---------A*X+B
                ---------Line space printed by Line 5Ø.
        ? 2
        ? 3
        ? 8
         19
        ? CTRL (^)/C
        STOP AT 1Ø
        >
```

4.7 Exercise

How would you modify the program so that, after typing RUN, you could supply one set of values for A and B, followed by several values of X?

See Appendix A for the answers.

4.8 Informative Printing

A program to print squares of numbers could print answers thusly:

```
        >RUN

        ? 23
         529

        ? 37
         1369

        ? 53
         2809

        ? 88
         7744

        ? and so forth
```

The following would be more preferable:

```
        >RUN

        COMPUTE X SQUARED

        WHAT IS X?  23
        X SQUARED = 529

        WHAT IS X?  37
        X SQUARED = 1369

        WHAT IS X?  53
        X SQUARED = 2809

        WHAT IS X?  88
        X SQUARED = 7744

        WHAT IS X?
```

...and so on until someone types CTRL/C.

This program identifies the desired input and the computed and printed output.

The following are the first two statements:

```
        10 PRINT "COMPUTE X SQUARED"
        20 PRINT ""
```

Line 10 causes NSC Tiny BASIC to print the message COMPUTE X SQUARED. Line 20 prints a Line Feed.

The two statements:

```
        30 PRINT "WHAT IS X?";   -------Note the semicolon.
        40 INPUT X
```

Cause NSC Tiny BASIC to type:

```
        WHAT IS X?
```

and wait for a value of X. The question mark is the prompt from the INPUT statement. Did you observe the semicolon at the end of the PRINT statement? It prevents a carriage return and line feed from occurring. If you don't use a semicolon the following would happen:

```
        30 PRINT "WHAT IS X"----------No semicolon.
        40 INPUT X
```

Without the semicolon, NSC Tiny BASIC types:

```
        WHAT IS X
        ?
```

For this program, remember to use the semicolon at the right end of the PRINT statement.

```
        50 PRINT X SQUARED =";   -------Semicolon.
        60 PRINT X*X
```

Together these two statements cause NSC Tiny BASIC to print the message "X SQUARED =" followed by the value of X*X. For example, if X = 23, NSC Tiny BASIC will type:

```
        X SQUARED = 529
```

Remember to note the semicolon on the right end of Line 50. Had it been omitted the following is what would happen:

```
        50 PRINT "X SQUARED =" --------No semicolon.
        60 PRINT X*X
```

If X = 23, NIBL will type

```
        X SQUARED =
        529
```

One more statement:

```
        70 GO TO 20
```

The following is everything put together in a complete program:

```
        10 PRINT "COMPUTE X SQUARED"
        20 PRINT ""
        30 PRINT "WHAT IS X";
        40 INPUT X
        50 PRINT "X SQUARED=";
        60 PRINT X*X
        70 GO TO 20
```

Load the above program into your INS8073 and RUN it. Try it for X = 23, 37, 53 and 88.

4.9 Multiple Statements Per Line

The following instructions explain how to put two or more statements on one line.

Instead of:

```
30 PRINT "WHAT IS X";
40 INPUT X
```

You can put both statements on one line:

```
30 PRINT "WHAT IS X"  :  INPUT X
(first statement)     ↑  (second statement)
                      |
 (The statements are separated by a colon)
```

To put more than two statements on a single line, follow the same format as above and be certain to separate each statement with a colon (:).

Instead of:

```
        20 PRINT ""
        30 PRINT "WHAT IS X";
        40 INPUT X
```

Put all three statements on one line:

```
20 PRINT "" : PRINT "WHAT IS X"; : INPUT X
            ↑                    ↑
   1st      |       2nd          |     3rd
statement   |    statement       |  statement
            |                    |
          colon                colon
```

The following is an example of four statements on one line.

```
        4Ø INPUT X: PRINT "X SQUARED=";: INPUT X : PRINT X*X : GO TO 2Ø

        Instead of:

        4Ø INPUT X
        5Ø PRINT "X SQUARED=";
        65 PRINT X*X
        7Ø GO TO 2Ø
```

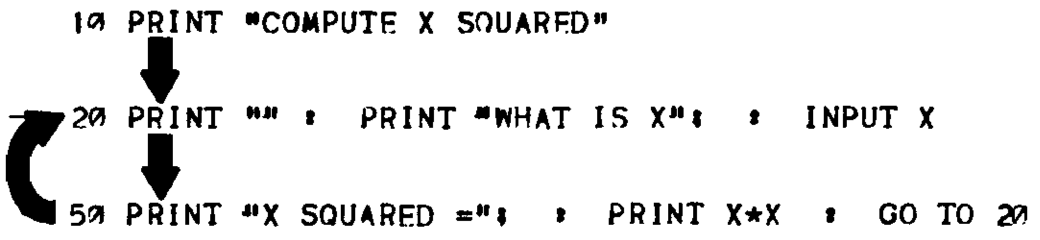
the following is a "compact" program to compute X^2, featuring the use of multiple statements per line:

```
        1Ø PRINT "COMPUTE X SQUARED"
        2Ø PRINT "" : PRINT "WHAT IS X"; : INPUT X
        5Ø PRINT "X SQUARED =";  : PRINT X*X : GO TO 2Ø
```

Try it on your INS8Ø73.

Follow the arrows to see how the program works.

```
        RUN
```



```
        1Ø PRINT "COMPUTE X SQUARED"

        2Ø PRINT ""  :  PRINT "WHAT IS X";  :  INPUT X

        5Ø PRINT "X SQUARED =";  :  PRINT X*X  :  GO TO 2Ø
```

As per standard, NSC Tiny BASIC does lines in line number order, first Line 1Ø, then to Line 2Ø, then Line 5Ø. NSC Tiny BASIC does all statements on a line in left to right order before moving on to the next line. Since Line 5Ø ends with a GO TO 2Ø statement, NSC Tiny BASIC, indeed, goes to Line 2Ø and continues, after finishing Line 5Ø.

In order to emphasize that multiple statements per line are separated by colons (:), a space on each side of the colon has been added; this is optional and Line 2Ø could have been typed:

```
        2Ø PRINT "":PRINT "WHAT IS X";:INPUT X
```

Some statements such as PRINT and INPUT can take multiple arguments. This allows several statements to be added together into one. For example:

```
      1ØPRINT X;:PRINT 4;:PRINT "DOMINO":INPUT A:INPUT B
```

can be shrunk to:

```
      1Ø PRINT X,Y, "DOMINO":INPUT A,B
```

4.1Ø Exercises

1. Write two programs to compute the value of AX+B for input values of A, X and B, as illustrated by the following RUN of our program.

```
        >RUN

        PROGRAM TO COMPUTE A*X+B

        A=?  2
        B=?  3

        X=?  5
        A*X+B = 13

        X=?  8
        A*X+B = 19

        X=? 12
        A*X+B = 27

        X=? ...and so on...press CTRL and C to abort program.
```

   A. Program No. 1. Do not use multiple statements per line.

   B. Program No. 2. Use multiple statements per line.

   Answers are in Appendix A

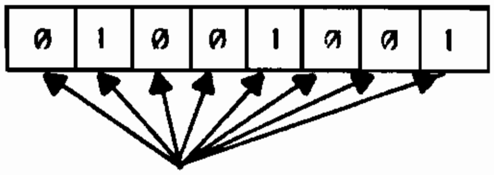
CHAPTER 5

5.1 Bits and Bytes

We assume that you are using an INS8Ø73 with at least 256 memory locations: this is the minimum configuration to run NSC Tiny BASIC.

- o Each memory location holds, or stores, one byte of information.
- o One byte consists of eight binary digits commonly called bits. BIT = BINARY DIGIT
- o One byte = 8 bits.
- o A binary digit (bit) is either Ø or 1.

You can think of a memory location as shown in the following diagram:



The number, 73, is stored in binary.

1 BYTE = 8 BITS = 1 MEMORY LOCATION

Each bit must be Ø or 1. Below are some numbers shown stored in bytes:

NUMBER (DECIMAL)	STORED AS A BYTE (BINARY)							
Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø	Ø
1	Ø	Ø	Ø	Ø	Ø	Ø	Ø	1
2	Ø	Ø	Ø	Ø	Ø	Ø	1	Ø
4	Ø	Ø	Ø	Ø	Ø	1	Ø	Ø
8	Ø	Ø	Ø	Ø	1	Ø	Ø	Ø
16	Ø	Ø	Ø	1	Ø	Ø	Ø	Ø
32	Ø	Ø	1	Ø	Ø	Ø	Ø	Ø
64	Ø	1	Ø	Ø	Ø	Ø	Ø	Ø
128	1	Ø	Ø	Ø	Ø	Ø	Ø	Ø

5.2 Exercises

Figure out how 3, 6, 7 and 29 would be stored. What is the largest that can be stored in one byte?

Answers are in Appendix A

5.3 Memory Address

Each memory location has a unique numeric address. The NSC Tiny BASIC program in the INS8Ø73 system occupies locations with addresses Ø to 2559.

An expanded INS8Ø73 system might have more memory locations. For example, your system may have 8192 locations, or 12288 locations... and so on, up to a maximum of 65535 locations, which includes "locations" that are really ports for peripheral devices.

Memory addresses might run from Ø to 4Ø95 or Ø to 8191, or Ø to 12287, and so on.

- o Memory locations Ø to 2559 hold NSC Tiny BASIC in the on-chip ROM (Read Only Memory) of the INS8Ø73.

- o Addresses 256Ø through 65471 are yours to use. When you type in an NSC Tiny BASIC program, you use some of these. The longer your program, the more you use. If you wire up some interesting electronic gadgets to the system, you will most likely use some of these addresses. Not all of these memory locations will actually be there in a typical system.

5.4 Hexadecimal Number System

To understand the literature, you are going to have to learn hexadecimal. The hexadecimal (base sixteen) number system is a handy shorthand for talking about bits and bytes and memory addresses.

In hexadecimal, addresses range from #ØØØØ to #FFFF.

The number sign (#) is used to tell you that the number is hexadecimal instead of decimal. This is the notation used in NSC Tiny BASIC: other notations exist in other literature.

This is a decimal number:	28673
This is a hexadecimal number:	#7ØØ1

The hexadecimal system has more digits than the decimal system.

Decimal digits:	Ø 1 2 3 4 5 6 7 8 9
Hexadecimal digits:	Ø 1 2 3 4 5 6 7 8 9 A B C D E F

Just as in the decimal system, each hexadecimal digit has a positional (or place) value. The digit occupying any position is multiplied by the value of that particular position. These products are then added together to obtain the value of the number.

Hexadecimal position values are expressed as powers of sixteen (rather than 10 as in the decimal system). Positions are numbered from right to left according to the increasing powers:

POSITION 3	POSITION 2	POSITION 1	POSITION 0
16^3	16^2	16^1	16^0

The decimal values of the powers of sixteen are:

$16^0 = 1 \quad 16^1 = 16 \quad 16^2 = 256 \quad 16^3 = 4096$

Check the decimal equivalents of the the following hexadecimal numbers.

$\#7001 = (7 \times 16^3) + (0 \times 16^2) + (0 \times 16^1) + (1 \times 16^0)$

$= (7 \times 4096) + 0 + 0 + 1$

$= 28672 + 0 + 0 + 1 = 28673$

$\#7002 = 28672 + 0 + 0 + 2 = 28674$

$\#7004 = 28672 + 0 + 0 + 4 = 28676$

$\#7010 = 28672 + 0 + 16 + 0 = 28688$

$\#7020 = 28672 + 0 + 32 + 0 = 28704$

(Remember, # in front of a number means it is hexadecimal.)

You will notice that in Section 3 a hexadecimal number is referred to by preceding the number with an "X'" instead of the "#" sign, for example:

X'8000

This is a more standard notation for hexadecimal numbers, but NSC Tiny BASIC does not like it.

If we ask the INS8Ø73 in NSC Tiny BASIC to print a hexadecimal number, NSC Tiny BASIC prints the decimal equivalent.

```
>PRINT #7ØØ1
 28673

>PRINT #A
 1Ø

>PRINT #B
 11

>PRINT #C
 12

>PRINT #D
 13

>PRINT #E
 14

>PRINT #F
 15

>PRINT #1Ø
 16

And so on...
```

The following is a table of hexadecimal digits vs. decimal values.

HEXADECIMAL DIGIT	DECIMAL VALUE
Ø	Ø
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
A	1Ø
B	11
C	12
D	13
E	14
F	15

You may wish to use the following small program for further experimentation:

```
>NEW

>100 REMARK HEXADECIMAL TO DECIMAL
>110 PRINT ""
>120 PRINT "HEXADECIMAL NUMBER"::INPUT H
>130 PRINT "DECIMAL EQUIVALENT IS",H - This is a multiple PRINT
>140 GO TO 110                          statement, see section
                                        4.8.
>RUN

 HEXADECIMAL NUMBER?  #7001
 DECIMAL EQUIVALENT IS 28673

 HEXADECIMAL NUMBER?...And so on.  If you type a decimal number
                                   (without #), you will get
                                   the decimal equivalent of
                                   your decimal number.
```

You may have noticed something new in Line 100. Any line that begins with the word "REMARK" is ignored by NSC Tiny BASIC, even if it contains another statement preceded by a colon. These REMARKS are used to help document the program; and, REMARK statements will be found in great abundance in the programs that follow in this primer.

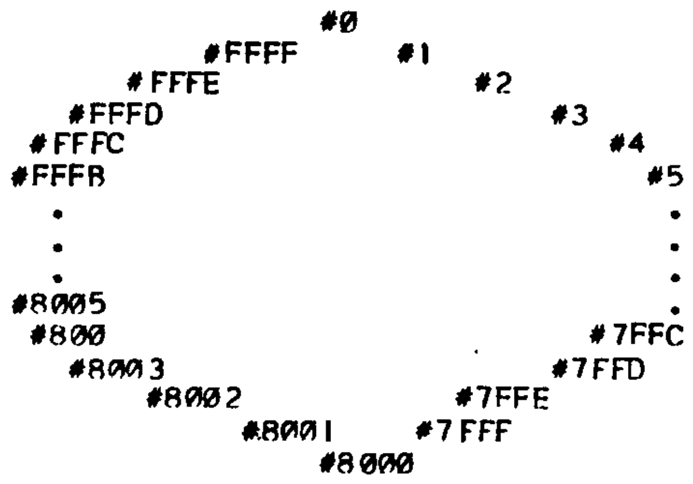
5.5 More About Hexadecimal

The hexadecimal numbers #0 to #7FFF, inclusive, are equivalent to the decimal numbers, 0 to 32767, inclusive. You can obtain the decimal equivalent of any hexadecimal number in the above range by using the program on this page.

To find out about the hexadecimal numbers from #8000 to #FFFF, use the program on this page. Enter that program and type RUN.

```
>RUN

HEXADECIMAL NUMBER? #8000
DECIMAL EQUIVALENT IS -32768

HEXADECIMAL NUMBER? #8001
DECIMAL EQUIVALENT IS -32767

HEXADECIMAL NUMBER? #FFFF
DECIMAL EQUIVALENT IS -1

HEXADECIMAL NUMBER? #FFFE
DECIMAL EQUIVALENT IS -2

And so on...
```

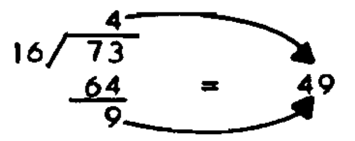
Remember the number circle in Chapter 3? It works in hexadecimal too:



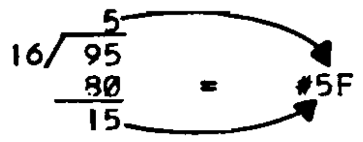
Compare the hexadecimal number circle with the decimal circle in Chapter 3. Below is a table showing some of the equivalences between decimal and hexadecimal NSC Tiny BASIC numbers:

POSITIVE NUMBERS		NEGATIVE NUMBERS	
Hexadecimal	Decimal	Hexadecimal	Decimal
#1	1	#8ØØØ	-32768
#2	2	#8ØØ1	-32767
#3	3	#8ØØ2	-32766
#4	4	#8ØØ3	-32765
•	•	•	•
•	•	•	•
#7FFD	32765	#FFFD	-3
#7FFE	32766	#FFFE	-2
#7FFF	32767	#FFFF	-1

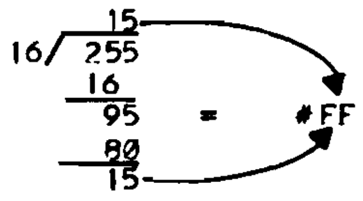
NSC Tiny BASIC automatically converts numbers from hexadecimal to decimal during print out: however, there is no built-in method for printing numbers directly in hexadecimal. The following examples illustrate the method used to convert decimal numbers Ø to 255 to hexadecimal:



Check: 4 x 16 + 9 = 73



Check: 5 x 16 + 15 = 95



Check: 15 x 16 + 15 = 255

#F = 15 #F = 15

You can convert any decimal number, Ø to 255, to hexadecimal as follows:

1. Divide the decimal number by 16, obtaining the quotient Q and remainder R.

2. For decimal numbers in the range Ø to 255, the quotient Q and the remainder R will each be numbers in the range Ø to 15, inclusive.

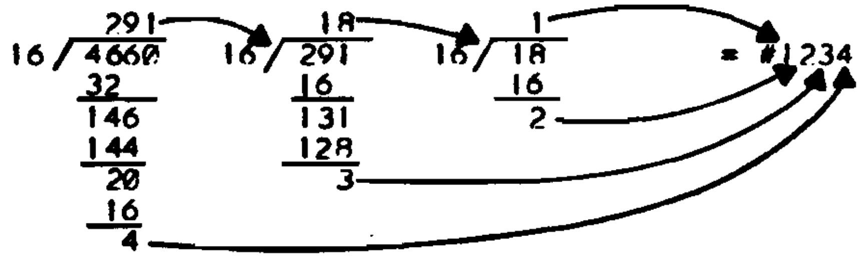
3. The hexadecimal number is #Q′R′ where Q′ and R′ are the hexadecimal digits (Ø through F) corresponding to the values of Q and R.

The following is a program to compute Q and R, this program features a new function, called MOD, for computing R.

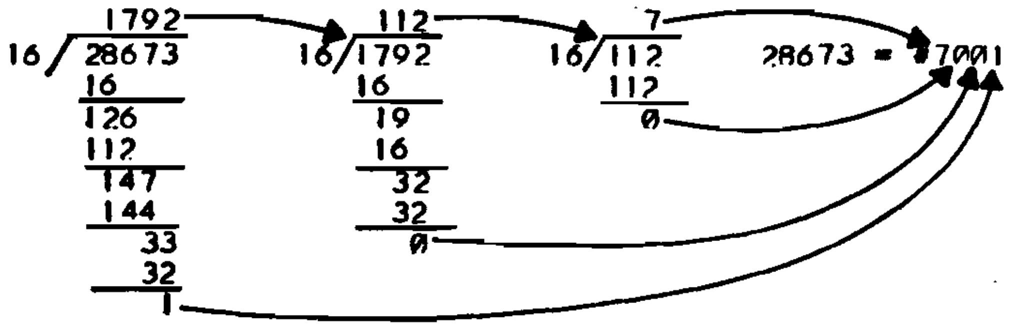
```
>NEW

>1ØØ REMARK CONVERT DECIMAL TO HEXADECIMAL, SORT OF
>11Ø PRINT "":PRINT "YOUR NUMBER"::INPUT N
>12Ø R=MOD(N,16)
>13Ø Q=N/16
>14Ø PRINT "HEXADECIMAL DIGIT VALUES:",Q,R
>15Ø GO TO 11Ø

>RUN

YOUR NUMBER? 73
HEXADECIMAL DIGIT VALUES:  4   9    Therefore, 73 = #49

YOUR NUMBER? 95
HEXADECIMAL DIGIT VALUES:  5  15    Therefore, 95 = #5F

YOUR NUMBER? 255
HEXADECIAL DIGIT VALUES:  15   5    Therefore, 255 = #FF
```

In Line 12Ø, the function MOD(N,16) computes the remainder on division of N by 16.

The following illustrates the method used to convert decimal numbers in the range Ø to 32767 to four digit hexadecimal numbers. Check it over very carefully:



The above is the conversion for 466Ø to #1234



The above is the conversion for 28673 to #7ØØ1

You try to convert 6844 to hexadecimal.

16/ 6844   16/   16/   6844 = #

(The check your conversion of 6844 to hexadecimal, look at the next program.)

The following program will work for numbers in the range Ø to 32767, inclusive.

```
>LIST

1ØØ REMARK CONVERT DECIMAL TO HEXADECIMAL, SORT OF
11Ø PRINT "":PRINT "YOUR NUMBER";:INPUT N
12Ø X=MOD(N,16)
13Ø N=N/16
14Ø W=MOD(N,16)
15Ø N=N/16
16Ø V=MOD(N,16)
17Ø U=N/16
18Ø PRINT "HEXADECIMAL DIGIT VALUES:"U,V,W,X
19Ø GO TO 11Ø

>RUN

YOUR NUMBER? 466Ø
HEXADECIMAL DIGIT VALUES: 1 2 3 4 ← 466Ø = #1234

YOUR NUMBER? 28683
HEXADECIMAL DIGIT VALUES: 7 Ø Ø 1 ← 28673 =  #7ØØ1

YOUR NUMBER? 6844
HEXADECIMAL DIGIT VALUES: 1 1Ø 11 12 ← 6844 = #1ABC

YOUR NUMBER? 255
HEXADECIMAL DIGIT VALUES: Ø Ø 15 15 ← 255 = #ØØFF = #FF
```

```
YOUR NUMBER? 32767
HEXADECIMAL DIGIT VALUES: 7 15 15 15←32767 = #7FFF

YOUR NUMBER? -1
HEXADECIMAL DIGIT VALUES: Ø Ø Ø -1←Beware of negative numbers!

YOUR NUMBER? -32767
HEXADECIMAL DIGIT VALUES: -7 -15 -15 -15

YOUR NUMBER? #7ØØ1
HEXADECIMAL DIGIT VALUES: 7 Ø Ø 1 - Hexadecimal is converted to
                                    hexadecimal, provided the
                                    number is in the range #Ø to
                                    #7FFF.

YOUR NUMBER #8ØØØ
HEXADECIMAL DIGIT VALUES? -8 Ø Ø Ø  Other hexadecimal numbers give
                                    funny results.  A complete ex-
                                    planation will not be attempt-
                                    ed in this primer.

YOUR NUMBER?         (No GO TO statement at end of program)
```

In case you haven't figured out how the program works, follow along as the program for N = 466Ø is traced. The following trace shows the values of variables after the statement on the same line has been executed.

STATEMENT	N	U	V	W	X
11Ø INPUT N	466Ø				
12Ø X=MOD(N,16)	466Ø				4
13Ø N=N/16	291				4
14Ø W=MOD(N,16)	291			3	4
15Ø N=N/16	18			3	4
16Ø V=MOD(N,16)	18		2	3	4
17Ø U=N/16	18	1	2	3	4
18Ø Prints the values of U,V,W, and X					
19Ø REPEAT THE PROGRAM AD INFINITUM					

5.6 Exercise

Trace the program for N = 6844:

STATEMENT	N	U	V	W	X
110 INPUT N	6844				
120 X = MOD (N,16)	___				___
130 N = N/16	___				___
140 W = MOD (N,16)	___			___	___
150 N = N/16	___			___	___
160 V = MOD (N,16)	___		___	___	___
170 U = N/16	___	___	___	___	___

Answers are in Appendix A

CHAPTER 6

6.1 The IF Statement

The useful and powerful IF statement permits programs to be written in which the computer makes simple decisions.

The following is an IF statement:

```
        IF P=14 THEN PRINT "AIR PRESSURE IS NORMAL"
```

This statement tells the computer "IF the value of P is equal to fourteen, then print the message "AIR PRESSURE IS NORMAL.". Not stated, but implied, is that IF P is not equal to fourteen, the message is not printed.

The following is an example of the IF statement used in a short program:

```
        100 REM AIR PRESSURE MONITOR
        110 PRINT "":PRINT "WHAT IS AIR PRESSURE"::INPUT P
        120 IF P = 14 THEN PRINT "AIR PRESSURE IS NORMAL"
        130 GO TO 110
```

You may have noticed that an abbreviated form of the "Remark" statement was used in Line 100. NSC Tiny BASIC only needs the first three letters to recognize the word: "REM" can be used as an abbreviation for the word REMARK.

Next, run the program and supply several values for air pressure, P.

```
        >RUN

        WHAT IS AIR PRESSURE?  14
        AIR PRESSURE IS NORMAL

        WHAT IS AIR PRESSURE?  14
        AIR PRESSURE IS NORMAL

        WHAT IS AIR PRESSURE?  23 ┐
                                  │
        WHAT IS AIR PRESSURE?  20 ├No message is printed
                                  │
        WHAT IS AIR PRESSURE?   0 ┘

        WHAT IS AIR PRESSURE? ^ C◀──(Control/C was pressed)
        STOP AT 110

        >
```

It would be better to have NSC Tiny BASIC print messages and ring bells when the air pressure is NOT normal. Replace Line 120 with the following IF statement:

```
        120 IF P <> 14 THEN PRINT "AIR PRESSURE IS NOT NORMAL bells"
```

In NSC Tiny BASIC, <> means...not equal to...

The following is the complete program:

```
        100 REM AIR PRESSURE MONITOR AND ALARM
        110 PRINT "":PRINT "WHAT IS AIR PRESSURE";:INPUT P
        120 IF P <> 14 THEN PRINT "AIR PRESSURE IS NOT NORMAL bells"
        130 GO TO 110



        >RUN

        WHAT IS AIR PRESSURE?  14

        WHAT IS AIR PRESSURE?  14

        WHAT IS AIR PRESSURE?  14

        WHAT IS AIR PRESSURE?  50   (Trouble!)
        AIR PRESSURE IS NOT NORMAL Bells

        WHAT IS AIR PRESSURE?  12
        AIR PRESSURE IS NOT NORMAL Bells

        And so on.
```

In a situation where air pressure was actually being monitored, Line 110 would be replaced with a a method for automatically acquiring the value of the air pressure P; probably by means of an analog to digital converter wired into the INS8073's memory. For now, however, you will simulate the acquisition of data by means of INPUT statements and concentrate on the structure of the program itself.

Requiring P to be exactly 14 is a tight control; loosen things up a little and let normal pressure be anything from 13 to 15, inclusive. You want a warning printed whenever P is less than 13 or greater than 15.

```
        100 REM AIR PRESSURE MONITOR AND ALARM
        110 PRINT "":PRINT "WHAT IS AIR PRESSURE";:INPUT P
        120 IF P < 13 THEN PRINT "AIR PRESSURE IS NOT NORMAL Bells"
        130 IF P > 15 THEN PRINT "AIR PRESSURE IS NOT NORMAL Bells"
        140 GO TO 110
```

If P is less than 13, Line 120 will cause a warning/alarm to be printed; and, if P is greater than 15, Line 130 will cause the message to be printed. If P is 13, 14 or 15, no message will occur.

```
>RUN

WHAT IS AIR PRESSURE?  14

WHAT IS AIR PRESSURE?  13

WHAT IS AIR PRESSURE?  15

WHAT IS AIR PRESSURE?  12◀—(Pressure is less than 13)
AIR PRESSURE IS NOT NORMAL Bells

WHAT IS AIR PRESSURE?  16◀—(Pressure is more than 15)
AIR PRESURE IS NOT NORMAL Bells

And so on...
```

6.2 Exercise

Modify the above program, with just two small changes, so that when air pressure is not normal NSC Tiny BASIC will tell you whether it is too high or too low. A RUN might look like the following, change the last program to do this. Answers are in Appendix A

```
>RUN

WHAT IS AIR PRESSURE?  14

WHAT IS AIR PRESSURE?  13

WHAT IS AIR PRESSURE?  15

WHAT IS AIR PRESSURE?  16
WARNING!  AIR PRESSURE TOO HIGH

WHAT IS AIR PRESSURE?  12
WARNING! AIR PRESSURE TOO LOW

And so on...
```

Since you are monitoring air pressure between limits, change the program to give yourself a little more flexibility in setting the limits:

```
100 REM AIR PRESSURE MONITOR AND ALARM
110 REM L=LOWER LIMIT, U=UPPER LIMIT FOR NORMAL PRESSURE
120 L=13
130 U=15
140 REM ACQUIRE ACTUAL AIR PRESSURE, P
150 PRINT "":PRINT "WHAT IS AIR PRESSURE";:INPUT P
160 REM IF P IS OUTSIDE NORMAL LIMITS, PRINT MESSAGE
170 IF P<L THEN PRINT "WARNING! AIR PRESSURE TOO LOW"
180 IF P>U THEN PRINT "WARNING! AIR PRESSURE TOO HIGH"
190 REM GO GET ANOTHER VALUE OF P
200 GO TO 150
```

Try the preceding program: then, change the lower limit (L) and upper limit (U) in Lines 12Ø and 13Ø and try the program again.

Also try this: Combine Lines 19Ø and 2ØØ as follows:

```
        19Ø GO TO 15Ø:REM GO GET ANOTHER VALUE OF P
```

Line 19Ø now contains two statements, a GO TO which tells NSC Tiny BASIC what to do, and a REM (remark) which tells you what is happening.

You may wish to change Lines 12Ø and 13Ø to INPUT statements. In that case, a RUN might look like the following:

```
        >RUN

        LOWER LIMIT FOR NORMAL AIR PRESSURE?  13
        UPPER LIMIT FOR NORMAL AIR PRESSURE?  15

        WHAT IS AIR PRESSURE?  14

        WHAT IS AIR PRESSURE?  13

        WHAT IS AIR PRESSURE?  16
        WARNING! AIR PRESSURE TOO HIGH

        WHAT IS AIR PRESSURE?  12
        WARNING! AIR PRESSURE TOO LOW

        And so on...
```

In general, the IF statement has the form of THEN:

```
        IF condition - THEN statement
```

For example, the following are two IF statements you've already seen:

$$\underbrace{\texttt{IF P = 14}}_{\text{Condition}}\ \texttt{THEN}\ \underbrace{\texttt{PRINT "AIR PRESSURE IS NORMAL"}}_{\text{Statement}}$$

$$\underbrace{\texttt{IF P<L}}_{\text{Condition}}\ \texttt{THEN}\ \underbrace{\texttt{PRINT "WARNING! AIR PRESSURE TOO LOW"}}_{\text{Statement}}$$

The following is an IF statement that you will be using soon.

$$\underbrace{\texttt{IF F>2Ø}}_{\text{Condition}}\ \texttt{THEN}\ \underbrace{\texttt{GO TO 51Ø}}_{\text{Statement}}$$

The condition is frequently a comparison between two quantities. Here is a handy table of comparisons that can be used in IF statements:

NIBL Symbol	Meaning	Math Symbol
=	Is equal to	$=$
<	Is less than	$<$
>	Is greater than	$>$
<=	Is less than or equal to	$\leq$
>=	Is greater than or equal to	$\geq$
<>	Is not equal to, i.e., greater or less than	$\neq$

The quantities being compared can be numbers, variables or algebraic expressions. The comparison can be TRUE or FALSE.

Below are comparisons and their truth values, TRUE or FALSE:

3 + 5 > 6 is TRUE, always.

If A = 8 and B = 30, then 4*A <= B is FALSE

If A = 8 and B = 32, then 4*A <= B is TRUE

If A = 8 and B = 40, then 4*A <= B is TRUE

If the comparison is TRUE, then the next statement on the same line as the IF is executed. It can be any kind of statement: A PRINT, a GO TO, another IF, or even those kinds of statements yet to be introduced. If the comparison is FALSE, then the statement following the comparison is ignored and the next highest numbered statement is executed.

```
IF P<L THEN PRINT "WARNING! AIR PRESSURE TOO LOW"
```

> Do this if the condition P < L is TRUE.
> Don't do this if P < L is FALSE.

That's all there is to IF statements, except that the word THEN may be omitted if you wish. For example, instead of writing:

```
IF P=14 THEN PRINT "AIR PRESSURE IS NORMAL"
```

you can omit the word THEN and write:

```
IF P=14 PRINT "AIR PRESSURE IS NORMAL"
```

Sometimes the word THEN makes the program easier to read. Use it if it feels comfortable.

Be careful to avoid making multiple statements separated by colons on a line with an IF statement. Remember that when an IF condition is found to be FALSE, the entire rest of the line is ignored. Therefore, for the following program, a zero will be printed.

```
        10 A=0:B=99
        20 IF B> 100 THEN PRINT "BIG B";A=1
        30 PRINT A
```

The following program has several REM's to help you read and understand it:

```
        100 REM DIALYSIS FLOW MONITOR PROGRAM

        110 REM GET FLOW RATE, F
        120 PRINT "":PRINT "FLOW";:INPUT F

        130 REM CHECK IF FLOW RATE CRITICALLY HIGH
        140 IF F>20 THEN GO TO 510

        150 REM CHECK IF FLOW RATE CRITICALLY LOW
        160 IF F<10 THEN GO TO 510

        170 REM CHECK IF FLOW RATE ABNORMALLY HIGH
        180 IF F>17 THEN GO TO 710

        190 REM CHECK IF FLOW RATE ABNORMALLY LOW
        200 IF F<13 THEN GO TO 710

        210 REM IF FLOW RATE IS NEITHER TOO HIGH NOR TOO LOW, IT IS OK
        220 PRINT "FLOW OK":GO TO 120

        500 REM FLOW RATE CRITICALLY HIGH OR LOW, SOUND BELLS
        510 PRINT "DANGER! FLOW RATE CRITICAL Bells:GO TO 120

        700 REM FLOW RATE IS ABNORMALLY HIGH OR LOW, PRINT MESSAGE
        710 PRINT "WARNING:  FLOW RATE ABNORMAL":GO TO 120
```

Try this program, make sure it works for all possible conditions. Try the following flow rates as test cases.

FLOW OK: 13, 14, 15, 16, 17

ABNORMAL: 10, 11, 12, 18, 19, 20

CRITICAL: 7, 8, 9, 21, 22, 23

After you have convinced yourself that this program works, read the following analysis of it.

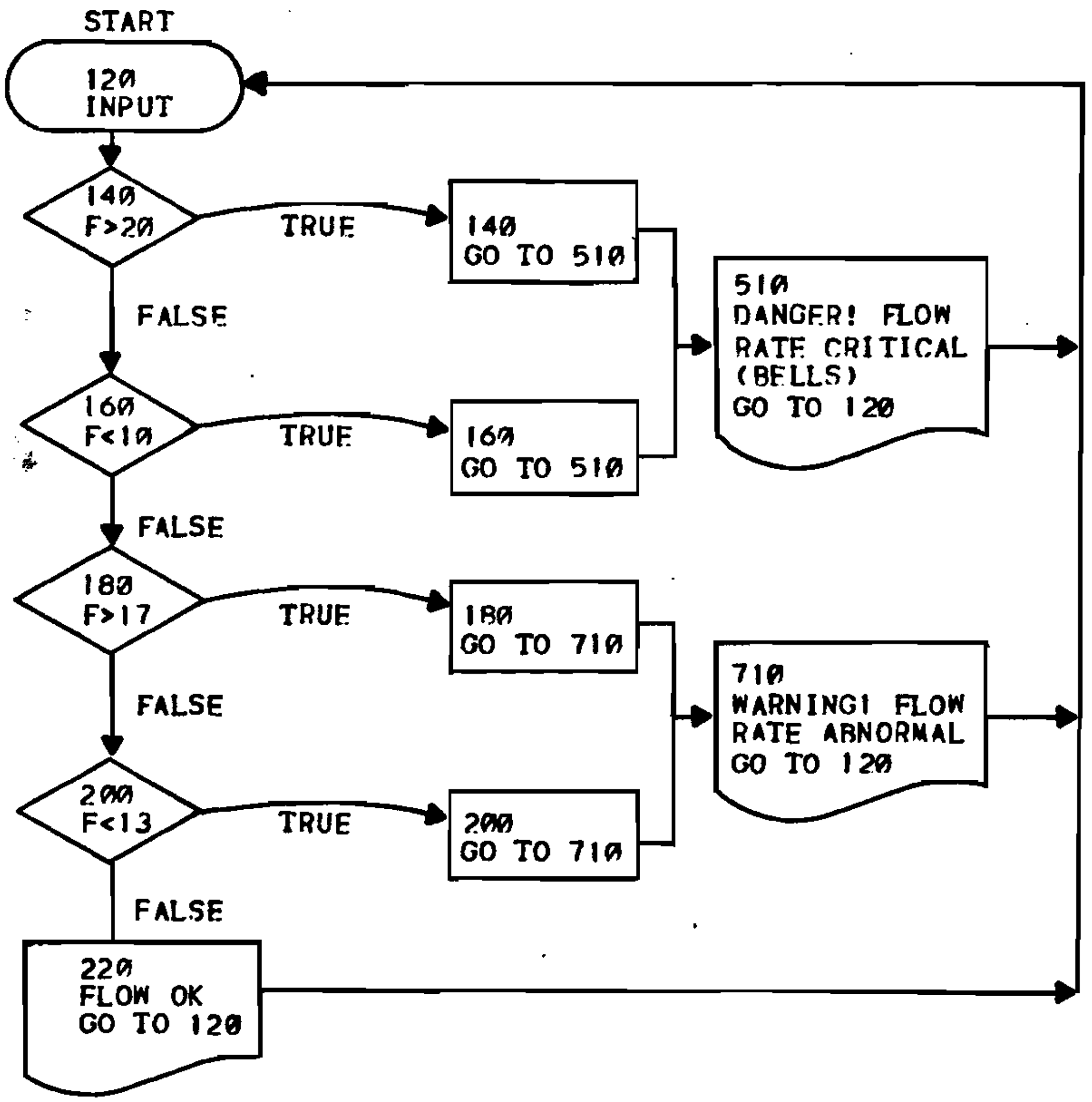
Follow along and trace through the program for a few specific values of F. First, suppose F = 25. The condition in Line 140 (F>20) is TRUE: therefore, NSC Tiny BASIC will go to 510. Line 510 directs NSC Tiny BASIC to print the message "DANGER! FLOW RATE CRITICAL", ring the TTY bell several times, then GO TO 120 for a new value of F. This will continue to happen for as long as F remains greater than 20.

Suppose F = 9. The condition in Line 140 (F>20) is FALSE, so NSC Tiny BASIC goes on to Line 160. In Line 160, the condition (F<10) is TRUE, so NSC Tiny BASIC will go to 510, print the danger message, ring the bell, then GO TO 120 for still another value of F.

Suppose F = 18. The condition in Lines 140 and 160 are both FALSE. (Check them yourself.) Therefore, NSC Tiny BASIC arrives at Line 180. The condition in Line 180 (F>17) is TRUE, so NSC Tiny BASIC does GO TO 710 and, as directed by Line 710 prints the message, "WARNING: FLOW RATE ABNORMAL", then goes back to Line 120 for another value of F.

The above has traced three possible paths through the program; there are two more, try these for F = 12 and F = 15. As there are five possible paths in all, you may wish to choose your favorite colors of felt tip pens and actually draw the paths.

Flowchart



In the flowchart, or logic diagram, of the Dialysis Flow Monitor Program, the diamond shaped boxes correspond to the IF statements. The numbers at the top of each box correspond to line numbers in the program. Compare the flowchart with the program. Trace through the flowchart for several values of F. Make sure you trace each of the five possible paths through the program. For example, try it for F = 25, 9, 18, 12 and 15. (Again, please note that it would be helpful to mark each path with a different color.)

6.3 A More Compact Program

In looking over the Dialysis Flow Monitor Program, we note the following:

1. If F>20 or F<10, the program should have a danger message plus alarm.

2. If the above is not true, and if F>17 or F<13, then the program should have an "abnormal" message, but not an alarm.

3. If neither of the above are true and everything is OK, a "FLOW OK" message will suffice.

NSC Tiny BASIC permits the use of logical operators AND, OR and NOT. Use is made of the OR operator in the following revision of the dialysis program.

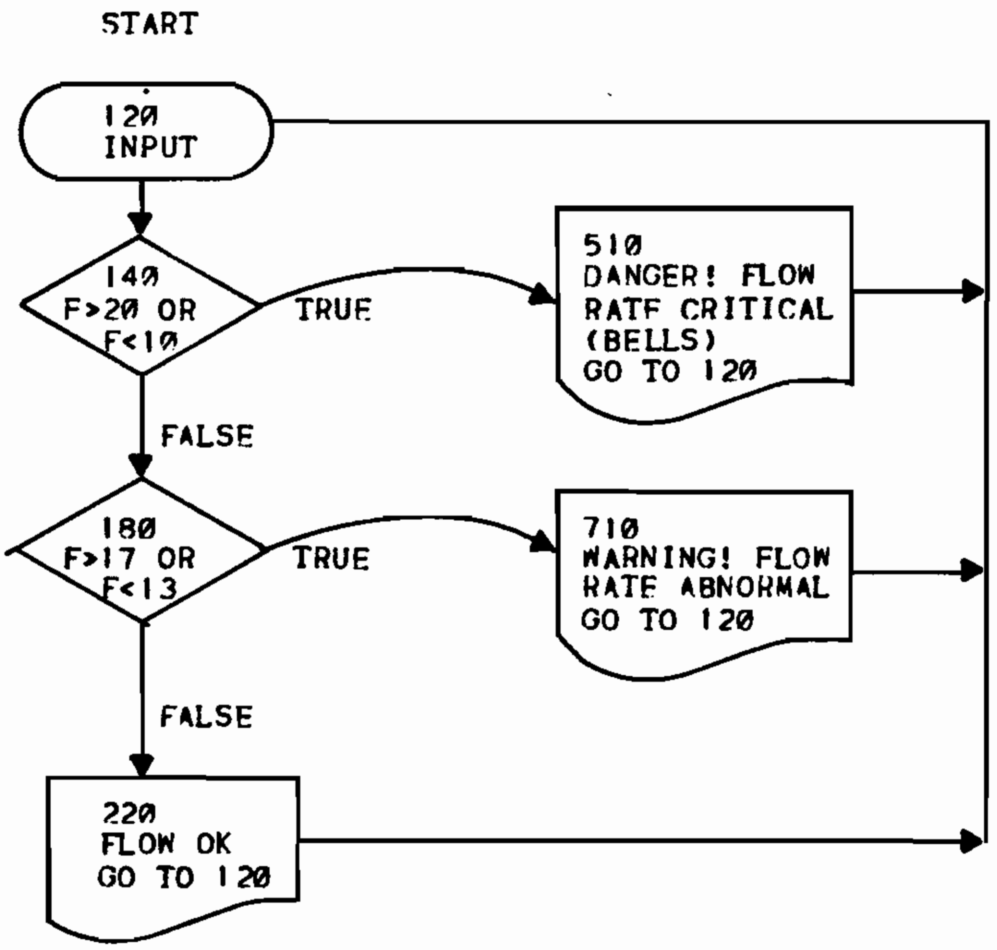
```
100 REM DIALYSIS FLOW MONITOR PROGRAM

110 REM GET FLOW RATE
120 PRINT "": PRINT "FLOW"::INPUT F

130 REM CHECK IF FLOW RATE CRITICALLY HIGH OR LOW
140 IF (F>20) OR (F<10) THEN GO TO 510

170 REM CHECK IF FLOW RATE ABNORMALLY HIGH OR LOW
180 IF (F>17) OR (F<13) THEN GO TO 710

210 REM IF FLOW RATE IS NEITHER TOO HIGH NOR TOO LOW, IT IS OK
220 PRINT "FLOW OK":GO TO 120

500 REM FLOW RATE CRITICALLY HIGH OR LOW, SOUND BELLS
510 PRINT "DANGER! FLOW RATE CRITICAL Bells:GO TO 120

700 REM FLOW RATE IS ABNORMALLY HIGH OR LOW, PRINT MESSAGE
710 PRINT "WARNING:  FLOW RATE ABNORMAL":GO TO 120
```

Suppose F=25. Then the compound condition (F>2Ø) OR (F<1Ø) in Line 14Ø is TRUE. In this case NSC Tiny BASIC will GO TO 51Ø. If F=9, the compound condition is also TRUE and NSC Tiny BASIC will GO TO 51Ø.

Suppose F=18. The compound condition (F>2Ø) OR (F<1Ø) in Line 14Ø is FALSE, so NSC Tiny BASIC continues on to Line 18Ø. Remember, F is now equal to 18, so the condition (F>17) OR (F<13) in Line 13Ø is TRUE. NSC Tiny BASIC does a GO TO 71Ø.

* * *The parentheses enclosing F>2Ø, F<1Ø and so on, are necessary! without them, the program will not work, because logical operators, as arithmetic operators, are evaluated from the left side of the expression to the right. Parentheses are used to give precedence.* * *

The following is a flowchart of the condensed dialysis program.


START
12Ø INPUT
14Ø F>2Ø OR F<1Ø
TRUE
51Ø DANGER! FLOW RATE CRITICAL (BELLS) GO TO 12Ø
FALSE
18Ø F>17 OR F<13
TRUE
71Ø WARNING! FLOW RATE ABNORMAL GO TO 12Ø
FALSE
22Ø FLOW OK GO TO 12Ø

Have you noticed that both programs tested for the most dangerous condition first? Then tested for the second most dangerous, simply as a matter of life-saving priorities. In this case, a few milliseconds probably won't make much difference; however, in many real time applications, a few milliseconds do make a difference.

To illustrate to you that programs usually can be improved upon, the following is a super-condensed Dialysis Flow Monitor Program:

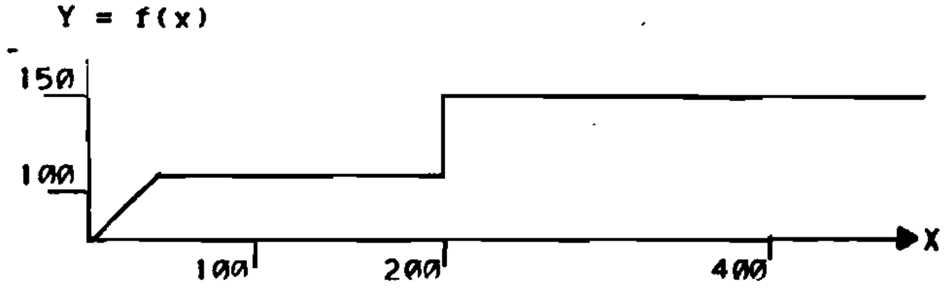
```
120 PRINT "":PRINT "FLOW";:INPUT F
140 IF (F>20) OR (F<10) PRINT "DANGER! FLOW RATE CRITICAL":
        GO TO 120
160 IF (F>17) OR (F<13) PRINT "WARNING:  FLOW RATE ABNORMAL":
        GO TO 120
220 PRINT "FLOW OK":GO TO 120
```

The AND, OR and NOT operators need not be limited to use in IF statements. They are logical operators and operate Bit-by-Bit on any constant or variable. This will be illustrated later on in this manual with an example on some I/O bits.

The following program implements the function indicated in the graph beneath it:

```
100 REM HASTILY CONSTRUCTED PROGRAM TO ILLUSTRATE USE OF "AND"
110 PRINT "":PRINT "X=";:INPUT X
120 IF (0<=X) AND (X<=100) PRINT "Y=",X:GO TO 110
130 IF (100<X) AND (X<=200) PRINT "Y=",100:GO TO 110
140 IF (200<X) AND (X<=400) PRINT "Y=",150:GO TO 110
150 PRINT "Y IS NOT DEFINED FOR X=",X
```

NOTE: The parentheses around 0<=X, X<=100, and so on in the IF statement are necessary. Without them, the program will not work. This is because of the multiplicity in the conditions being checked.



X for 0 < X < 100

y = f(x) = 100 for 100 < X < 200

150 for 200 < X < 400

The following is a RUN of the preceding program. All critical points have been checked.

```
RUN

X=? -1
Y IS NOT DEFINED FOR X=-1

X=? 0
Y= 0

X=? 37
Y= 37

X=? 99
Y= 99

X=? 100
Y= 100

X=? 101
Y= 101

X=? 199
Y= 100

X=? 200
Y= 100

X=? 201
Y= 150

X=? 299
Y= 150

X=? 300
Y= 150

X=? 301
Y- 150

X=? 399
Y- 150

X=? 400
Y- 150

X=? 401
Y IS NOT DEFINED FOR X = 401
```

And so on...

6.4 Random Numbers and Computer Games

Another useful feature in NSC Tiny BASIC is a random generator. Sometimes it is useful to generate random numbers between specific limits. A trivial use is to imitate a pair of dice. The statement:

```
   D = RND(1,6)
```

will make D some number between 1 and 6 inclusive, with equal probability for each of the possibilities. The following program simulates a pair of dice:

```
   10 PRINT RND(1,6), RND(1,6)
   20 GO TO 10
```

RUN the program for a while:

```
   >RUN
   4 1
   2 3
   3 5
   1 1
   3 2
   3 1
   4 2
   5 5
   3 5
   3 4
   1 4
   3 3
   1 6
   3 3
   3 2

   STOP AT 10
   >
```

In general, the expression:

```
   RND(A,B)
```

is a random integer between A and B, inclusive. A and B may be algebraic expressions, simple variables or constants. RND may be used wherever a variable may be used.

Random numbers are widely used to test programs, and to do Monte Carlo method solutions to problems. Many games use a random number generator.

```
 10 REM GUESS THE NUMBER GAME
 20 X=RND (1,100):REM X IS THE SECRET NUMBER FROM 1 TO 100
 30 PRINT"":PRINT "WHAT IS YOUR GUESS";
 40 INPUT G:REM G WILL BE THE GUESS
 50 IF G<X THEN PRINT "YOUR GUESS IS TOO SMALL"
 60 IF G>X THEN PRINT "YOUR GUESS IS TOO BIG"
 70 IG G=X THEN GO TO 90
 80 GO TO 30:REM NOT A CORRECT GUESS. GET NEXT GUESS
 90 PRINT "YOU WIN. LET'S PLAY AGAIN."
100 GO TO 20:REM GET A NEW SECRET NUMBER

>RUN

WHAT IS YOUR GUESS? 50
YOUR GUESS IS TOO BIG

WHAT IS YOUR GUESS? 25
YOUR GUESS IS TOO BIG

WHAT IS YOUR GUESS? 12
YOUR GUESS IS TOO SMALL

WHAT IS YOUR GUESS? 18
YOUR GUESS IS TOO SMALL

WHAT IS YOUR GUESS? 24
YOU WIN. LET'S PLAY AGAIN.
```

And so on...

6.5 Exercise

Rewrite Line 70 to combine the functions of Lines 70 and 80 making the program one statement shorter.

Answers are in Appendix A

CHAPTER 7

7.1 Program Loops

This section of the primer deals with Program Loops. The following program causes NSC Tiny BASIC to print out the first ten positive integers and the squares of those integers. While not exactly intriguing in its mathematical subtlety, it helps point out a few useful techniques in programming.

The following is an example of a cumbersome way to achieve the results described above:

```
>PRINT 1
 1

>PRINT 1*1
 1

>PRINT 2
 2

>PRINT 2*2
 4

>PRINT 3
 3

>PRINT 3*3
 9
```

And so on until...

```
>PRINT 10
 10

>PRINT 10*10
 100

>
```

The foregoing would get the results, interspersed with PRINT statements; or, a program could be written as follows:

```
10 REM PRINT THE FIRST TEN NUMBERS AND THEIR SQUARES
20 PRINT 1
30 PRINT 1*1
40 PRINT 2
50 PRINT 2*2
60 PRINT 3
70 PRINT 3*3
```

And so on until...

```
180 PRINT 9
190 PRINT 9*9
200 PRINT 10
210 PRINT 10*10
```

RUN the program: the following is what your RUN should look like:

```
>RUN
 1
 1
 2
 4
 3
 9
And so on until
 9
 81
 10
 100

>
```

Not a very readable chart, is it? Results that are hard to read or interpret decrease the value of the output. The answer must be communicated to those who need the results. By using a comma to keep the number and its square on the same line, and by using a PRINT statement you can write a much improved program.

Note, in the following, the use of a comma in PRINT statements to separate the number and the number squared:

```
10 REM TABLE OF NUMBERS AND THEIR SQUARES
20 PRINT " N N SQUARED "
30 PRINT 1,1*1
40 PRINT 2,2*2
50 PRINT 3,3*3
60 PRINT 4,4*4
70 PRINT 5,5*5
80 PRINT 6,6*6
90 PRINT 7,7*7
100 PRINT 8,8*8
110 PRINT 9,9*9
120 PRINT 10,10*10
```

If you store the above program in the INS8073's memory and RUN it, the results would be:

```
>RUN
 N N SQUARED
 1 1
 2 4
 3 9
 4 16
 5 25
 6 36
 7 49
 8 64
 9 81
 10 100
>
```

The above program is much easier to read than the first two presented in this chapter. Each number is printed side by side with its square in the order they appear in the PRINT statement. For example:

The statement, PRINT 7,7*7

Causes NSC Tiny BASIC to print, 7 49

You now have enough tools to write a very short program to print numbers and their squares. The idea is to write short programs that do a lot of work. Read the following program, and then try it on your computer. Type in all of the REMarks as they will help to explain what is happening. Remember, REMarks are for people: the computer simply ignores them.

```
10 REM A PROGRAM TO PRINT SUCCESSIVE INTEGERS AND THEIR SQUARES

15 REM PRINT A HEADING

17 PRINT " I I SQUARED"

20 REM USE A VARIABLE, I, TO HOLD THE VALUE OF THE NUMBER

30 REM TO BE SQUARED. START THE VALUE AT ONE

40 I=1

50 REM NOW THAT I HAS A VALUE, PRINT IT AND ITS SQUARE

60 PRINT I, I*I

70 REM ADD ONE TO THE VALUE OF I, TO CREATE THE NEXT LARGER

80 REM   INTEGER, SO THAT IT AND ITS SQUARE CAN BE PRINTED

90 REM   UP IN LINE 60

100 I=I+1

110 REM NOW THAT THE VALUE OF I IS ONE LARGER, GO TO LINE 60

120 GO TO 60
```

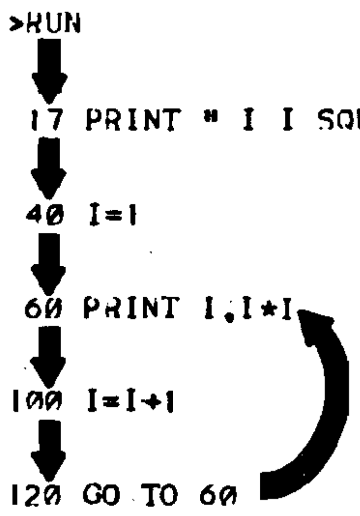
After you understand how it works, type in the program (or at least this abbreviated form) without the REMarks.

Try the following short form of the program on your INS8073:

```
    17 PRINT " I I SQUARED"
    40 I=1
    60 PRINT I,I*I
    100 I=I+1
    120 GO TO 60
```

Do you see what is going to happen? Did you remember to clear out any old program with NEW?

Do you have the program figured out? If not, follow the arrows:



```
>RUN
 17 PRINT " I I SQUARED"
 40 I=1
 60 PRINT I,I*I
100 I=I+1
120 GO TO 60
```

Lines 17 and 40 are done once.

Lines 60, 100 and 120 are "in the 'loop'". They are repeated again and again ...(until you press the BREAK key).

7.2 Exercises

1. If Line 17 is changed to read 17 PRINT " N N SQUARED" how would this change the results?

2. When you RUN the program, does NSC Tiny BASIC automatically stop after printing the first ten positive integers and their squares?

3. What is the largest value of I for which the program will give the correct answers?

Answers are in Appendix A

7.3 IF Loops

The program does not satisfy the initial requirements, that is, to print the squares of the first ten positive integers. Agreed, it does print the ten positive integers and their squares; but then it just keeps on going. You want it to stop automatically after printing 10 and 10 squared. The IF statement will help you to achieve your goal.

```
Instead of        120 GO TO 60

Use               120 IF I < 11 GO TO 60
```

```
 17 PRINT " I I SQUARED"
 40 I=1
 60 PRINT I,I*I
100 I=I+1
120 IF I<11 GO TO 60
```

The IF statement (Line 120) can be read: "If I is less than eleven then GO TO Line 60." Not stated, but implied, is that if I is not less than 11, in particular if it is 11 or more, then DO NOT GO to Line 60, but just go on to the next line. There is no next line, so the program will stop.

To make the program more complete, add the STOP statement. This statement, when executed, stops the program. Of course, as you have seen, the program stops if there is nothing else to do. Occasionally it is necessary to deliberately stop a program. It is also useful to put a STOP statement at the end of a program just to mark the end of that program. Add a STOP statement to the end of the program to compute squares.

```
 17 PRINT " I I SQUARED"
 40 PRINT I=1
 60 PRINT I,I*I
100 I=I+1
120 IF I<11 GO TO 60
999 STOP     <------------------------The STOP statement.
```

Any line number from 121 to 32767 could have been used for the STOP statement, 999 was arbitrarily chosen. It is often used to save the programmer's having to retype the entire "STOP" statement if he wants to add to the bottom of the existing program. The following is a RUN of the above program.

```
>RUN

 I I SQUARED
 1 1
 2 4
 3 9              NSC Tiny BASIC computed and printed I and I squared
 4 16             for I=1, 2, 3...10 and then stopped automatically.
 5 25
 6 36
 7 49
 8 64
 9 81
10 100
STOP At 999
```

7.4 Exercises

1. What will happen if you change Line 120 to 120 IF I < = 10 GO TO 60 and RUN the program again? (Try it on your system.)

2. What will happen if you change Line 120 to 120 IF I < 17 GO TO 60 and RUN the program again?

3. What will happen if you mistype Line 120 as 120 IF I < 11 GO TO 40 and ran the program again?

4. What would be the results of RUNning the following program?

```
        10 I=1
        20 PRINT I,I*I:I=I+1:IF I<11 GO TO 20
        99 STOP
```

For answers, see Appendix A

7.5 FOR NEXT Loops

When a program contains a statement that is executed more than once, then that program contains a LOOP. Nearly all the programs in this book contains loops. In fact, it is the loop that makes programming so powerful. If each statement could only be used once, then programming would be exceedingly tedious. As has been seen, programmers tend to write statements that can be used repeatedly rather than only once.

The very simple loop:

```
        10 PRINT 4
        20 GO TO 10
```

will run and print 4's indefinitely. Most loops have some facility for ending gracefully. What does the following program do?

```
        10 I=0
        20 I=I+1
        30 PRINT I;
        40 IF I < 10 THEN GO TO 20
```

The following program does the same thing:

```
        10 I=1
        20 PRINT I;
        30 I=I+1
        40 IF I < 11 THEN GO TO 20
```

Loops are so common that NSC Tiny BASIC provides a shorthand for writing them. The next program does exactly the same thing as the previous two:

```
        10 FOR I=1 TO 10 STEP 1─┐
        20 PRINT I;             ├---This is a FOR NEXT loop.
        30 NEXT I              ─┘
```

RUN the FOR NEXT loop:

```
        >RUN
         1 2 3 4 5 6 7 8 9 10     The FOR NEXT loop caused NSC Tiny
        >                         BASIC to print values of I for:
                                        I equals one (1) to
                                        I equals ten (10)
                                  in steps, or increments, of one.
```

The numbers go across the page instead of down because the PRINT statement ends with a semicolon (;). Try the program with a PRINT statement that doesn't end with a semicolon and you will get the following:

```
       >RUN
        1
        2
        3
        4
        5
        6
        7
        8
        9
        10
       >
```

The FOR statement sets up the loop. It specifies what the variable (often called the control variable for the loop) is to have for its initial value, then the final value and finally how much it is to be incremented each time through the loop.

The NEXT statement is the bottom of the loop, and says to find the next value of the control variable and continue execution at the statement immediately AFTER the corresponding FOR statement, if the control variable has not yet passed the final value.

To print the odd numbers from 1 to 10 (obviously 10 itself will not be one of them) the following loop could be used.

```
        10 FOR I=1 TO 10 STEP 2
        20 PRINT I;  <──────────────Semicolon causes numbers to be printed
        30 NEXT I                   across the line.

         1 3 5 7 9  <───────────────NSC Tiny BASIC is STEPping by 2.
        >
```

The step size can be negative:

```
        10 FOR I=56 TO 42 STEP -3
        20 PRINT I;
        30 NEXT I
```

Before running this program, figure what its output should be. The rule is: the FOR NEXT loop always starts exactly at the first value, and will not go beyond the second. Each time through the loop the STEP is added to the index. In these simple programs the variable I has been the index. (Of course, if the STEP is negative, adding it to the index makes the index smaller.)

The last program prints:

```
        56 53 50 47 44
```

You don't always have to use I. You can use any variable in a FOR statement as long as you use the same variable in the corresponding NEXT statement.

```
        10 FOR K=1 TO 3 STEP 1
        20 PRINT "HIP HIP HOORAY"
        30 NEXT K

        >RUN
        HIP HIP HOORAY
        HIP HIP HOORAY
        HIP HIP HOORAY

        >
```

If the STEP size is one, the STEP clause can be omitted.

```
        10 FOR A=0 TO 7
        20 PRINT A;
        30 NEXT A

        >RUN
         0 1 2 3 4 5 6 7
        >
```

The FOR NEXT loop makes it easy to run off tables, such as the table of I and I SQUARED.

```
        10 PRINT " I I SQUARED"
        20 FOR I=1 TO 5    ---------Since STEP size is 1, it's omitted.
        30 PRINT I, I*I
        40 NEXT I
```

```
>RUN
 I  I SQUARED
 1  1
 2  4
 3  9
 4  16
 5  25

>
```

To get a table for I-1, 2, 3,...,10, change Line 20 to: 20 FOR I=1 TO 10

To get a table running from 100 to 120 change Line 20 to: 20 FOR I=100 TO 120

Additionally, the starting and ending values can be variables or expressions. Here are two examples:

```
10 A=1
20 B=5
30 FOR X=A TO B
40 PRINT X;
50 NEXT X

>RUN
 1 2 3 4 5
>
```

```
10 L=2
20 U=3
30 FOR S=L*L TO U*U
40 PRINT S;
50 NEXT S

>RUN
 4 5 6 7 8 9
>
```

The STEP can also be a variable or an expression:

```
10 A=1
20 B=13
30 C=2
40 FOR X=A TO B STEP C
50 PRINT X;
60 NEXT X

>RUN
 1 3 5 7 9 11 13
>
```

Although the applications may not be readily apparant, FOR NEXT loops may be "nested" up to four levels. This means that you can have four layers of loops within loops. An example of this is shown below. The loops interact to print the numbers 0 to 99 in a square grid.

```
10 REM SQUARE MATRIX GRID. NUMBERS 1 TO 9 ARE PRINTED
20 REM ACROSS THE PAGE, THEN A CARRIAGE RETURN,
30 REM THEN NUMBERS 10 TO 19 ETC.
40 FOR I=0 TO 90 STEP 10:REM TENS LOOP
50 FOR J=0 TO 9:REM UNITS LOOP
60 PRINT I + J;:REM PRINT ACROSS PAGE
70 NEXT J:REM END OF "INNER" LOOP
80 PRINT:REM CARRIAGE RETURN
90 NEXT I:REM END OF OUTER LOOP
999 STOP
```

```
>RUN

Ø   1   2   3   4   5   6   7   8   9
1Ø  11  12  13  14  15  16  17  18  19
2Ø  21  22  23  24  25  26  27  28  29
3Ø  31  32  33  34  35  36  37  38  38
4Ø  41  42  43  44  45  46  47  48  49
5Ø  51  52  53  54  55  56  57  58  59
6Ø  61  62  63  64  65  66  67  68  69
7Ø  71  72  73  74  75  76  77  78  79
8Ø  81  82  83  84  85  86  87  88  89
9Ø  91  92  93  94  95  96  97  98  99

>
```

Loops cannot cross each other for obvious reasons: therefore, if you write:

```
1Ø FOR I=1 TO 1Ø
2Ø FOR J=1 TO 1Ø
3Ø NEXT I
4Ø NEXT J
```

you will get a FOR NEXT error message (ERROR 1Ø) upon execution of Line 3Ø.

7.6 Exercises

For each program, first figure out what you think NSC Tiny BASIC will do, then RUN the program and verify your thinking.

1.
```
1Ø S=Ø
2Ø FOR K=1 TO 5
3Ø S=S+K
4Ø PRINT K,S
5Ø NEXT K
```

2.
```
1Ø P=1
2Ø FOR K=1 TO 5
3Ø P=P*K
4Ø PRINT K,P
5Ø NEXT K
```

3.
```
1Ø S-Ø
2Ø FOR K=1 TO 5
3Ø S=S+K
4Ø NEXT K
5Ø PRINT S
```

4.
```
1Ø p=1
2Ø FOR K=1 TO 5
3Ø P=P*K
4Ø NEXT K
5Ø PRINT P
```

Each of the following programs requires an INPUT value for the variable, N. For the values outguess NSC Tiny BASIC then verify your results.

5.
```
1Ø INPUT N
2Ø S=Ø
3Ø FOR K=1 TO N
4Ø S=S+K
5Ø NEXT K
6Ø PRINT N,S
```

Try it for N=7

6.
```
1Ø INPUT N
2Ø P=1
3Ø FOR K=1 TO N
4Ø P=P*K
5Ø NEXT K
6Ø PRINT N,P
```

Try it for N=7 and N=9

7. Write a short program to compute and print the value of:

$$1^2 + 2^2 + 3^2 + \ldots + N^2$$

for an INPUT value of N.

```
>RUN

 N=? 5
 SUMSQUARED = 55

>
```

Answers are in Appendix A

7.7 The DO Statement

This is a simple statement that instructs NSC Tiny BASIC to DO a function UNTIL another condition is met. It has only one form:

```
>340 DO
```

Nothing ever comes after the word "DO"; somewhere later on in the program there is a statement UNTIL, for example:

```
>500 UNTIL (some arithmetic instruction goes here)
```

The line numbers are only examples. The UNTIL statement acts just like a GO TO which causes execution to proceed from the DO statement - whenever the value of the arithmetic expression equals zero. The following program:

```
>10 DO
>20 PRINT "HELLO"
>25 N=1
>30 UNTIL N=0
```

prints the word HELLO over and over until you stop the program.

If instead, you had said:

```
>10 N=5
>20 DO
>30 PRINT "THIS ONE STOPS SOON"
>40 N=N-1
>50 UNTIL N=0
```

the message is printed out five times. It is interesting to try this program with Line 30 changed to PRINT N. To understand how it works you must remember that an expression such as N=0 has a numerical value. It is 0 as long as it is false. Thus, when N is not zero it is false to say that N=0. So the expression N=0 has the numerical value 0. The UNTIL acts like a GO TO back to Line 20, but when N is indeed zero then the expression N=0 is true. When an expression is true its value is not zero, but -1. So the UNTIL does not act like a GO TO and the program ends.

NSC Tiny BASIC uses the notation:

```
X=6
```

in two different ways. If it is a statement, where the X is the first item on the line it means "let the value of X be 6" but if the notation X=6 is used as part of an expression - that is, not as the first item on the line - then it means "see if X is equal to six" and if X is equal to six then the whole expression (X=6) takes on the value -1. This is similar to the test in an IF statement.

The converse of this is also usable, that is, a DO...UNTIL loop can terminate in any of the following with the same effect:

```
UNTIL K=N
UNTIL K-N=0
UNTIL 0=K-N
UNTIL K-N  (which automatically has a value of zero allowing
            the loop to be terminated)
```

For reasons of clarity, only the first and third examples should be used. In the following:

```
>10 G=3
>20 G=G=3
>30 PRINT G
```

(would be a very confusing thing to write and not at all recommended) the value -1 would be printed, as Line 20 would make the value of G equal to (G-3) which is true, and thus has the value -1.

For the most part, the ambiguity mentioned can be safely ignored, and the UNTIL statement taken at "face value" where you know that the loop will be done over and over again until the indicated condition is satisfied. The pair of statements:

```
DO
.
.
.
UNTIL K > (N/2)
```

will execute whatever is between them until it happens that K is greater than half of N. (Remember that NSC Tiny BASIC only does integer arithmetic so that N/2 means the integer part only, any remainder or fractional part is ignored.) DO UNTIL loops, like FOR NEXT loops, can be nested, but where FOR NEXT loops can go four deep, (four layers of of loops) DO UNTIL is allowed eight levels of nesting.

7.8 Powers Of Two

The following are three programs to compute and print the powers of 2 from 1 to 64, inclusive.

Program number one uses an IF loop.

```
        100 REMARK POWERS OF TWO, PROGRAM ONE
        110 P=1
        120 PRINT P;
        130 P=2*P
        140 IF P<=64 GO TO 120
        999 STOP
```

Program number two uses a FOR NEXT loop.

```
        100 REMARK POWERS OF TWO, PROGRAM TWO
        110 P=1
        120 FOR K=0 TO 6
        130 PRINT P;
        140 P=2*P
        150 NEXT K
        999 STOP
```

Program number three uses a DO UNTIL loop.

```
        100 REMARK POWERS OF TWO, PROGRAM THREE
        110 P=1
        120 DO
        130 PRINT P;
        140 P=2*P
        150 UNTIL P>64   - - - (Also try UNTIL P=128 here.)
        999 STOP
```

These three programs produce exactly the same results. If you enter any of the above and type RUN, here is what happens:

```
        >RUN
         1 2 4 8 16 32 64
        >
```

Try to modify each program to get powers of 2 from 1 to 128. Then do it backwards, get powers of two from 64 to 1.

IF Loop

```
        100 REMARK POWERS OF TWO, PROGRAM FOUR
        110 P=64
        120 PRINT P;
        130 P=P/2
        140 IF P<1 GO TO120
        999 STOP
```

FOR NEXT Loop

```
        100 REMARK POWERS OF TWO, PROGRAM FIVE
        110 P=64
        120 FOR K=6 TO 0 STEP -1 ———(or FOR K=0 TO 6)
        130 PRINT P;
        140 P=P/2
        150 NEXT K
        999 STOP
```

DO UNTIL Loop

```
        100 REMARK POWERS OF TWO, PROGRAM SIX
        110 P=64
        120 DO
        130 PRINT P;
        140 P=P/2
        150 UNTIL P<1 ----(Also try UNTIL P=0 here)
        999 STOP
```

The three programs all produce the same results.

```
        >RUN
         64  32  16  8  4  2  1
        >
```

Modify each program so that NSC Tiny BASIC types powers of 2 from 128 to 1.

CHAPTER 8

8.1 Subroutines

As you learn to program, you will find that your programs will increase in size until they become unmanageable. When that happens, it's time to break them up into functional blocks. Often, you will find that some of those functional blocks are used in several places in your program. Rather than repeat them each time, a "subroutine" can be used. A subroutine is just a section of NSC Tiny BASIC statements performing some operation required at more than one place in the program. The GOSUB statement is used to transfer control to the subroutine and the RETURN statement is used to return control to the place where the subroutine was called.

As with FOR and NEXT, and the DO and UNTIL statements, GOSUB and RETURN are a pair that are always together. The statement:

```
        125 GOSUB 910
```

acts exactly like the statement:

```
        125 GO TO 910
```

except that the computer remembers the line number (in this case 125) of the statement that made it go to Line 910, so that after the subroutine is finished the computer can resume where it left off. For example, when the program executes a RETURN statement:

```
        320 RETURN
```

the computer knows to jump back to the statement immediately after Line 125 - wherever it is.

The following is a short program to demonstrate the use of the GOSUB and RETURN statements. Try it on your INS8073.

```
        100 N=1      ---------┐
        105 GOSUB 200         ┤
        110 N=2               ┤
        115 GOSUB 200         ┤
        120 N=3               ┤
        125 GOSUB 200         ┤
        130 N=4               ┤     Main Program
        135 GOSUB 200         ┤
        140 N=2               ┤
        145 GOSUB 200         ┤
        150 N=3               ┤
        155 GOSUB 200         ┤
        160 END      ---------┘

        200 PRINT "THIS IS NUMBER ",N┐---------Subroutine
        210 RETURN                   ┘
```

RUN the above program, the results should look like this:

```
        >RUN

         THIS IS NUMBER 1
         THIS IS NUMBER 2
         THIS IS NUMBER 3
         THIS IS NUMBER 4
         THIS IS NUMBER 5
         THIS IS NUMBER 6

        >
```

The statement:	110 GOSUB 200
tells NSC Tiny BASIC:	GO TO line 200, but remember that you came from Line 110. When you come to a RETURN statement, return to the line next after Line 110 (which is Line 115).
The statement:	120 GOSUB 200
tells NSC Tiny BASIC:	GO TO Line 200, but remember that you came from Line 120. When you come to a RETURN statement, return to the line next after Line 120.

The rest of the GOSUBs operate in a similar fashion. GOSUB, when used properly, can eliminate the tedium of having to retype a routine wherever it may be used in a large program. Subroutines may call other subroutines, this is called nesting and may look like this:

```
        10 I=12
        20 GOSUB 200
        30 I=5
        40 GOSUB 200
        50 A=2
        60 B=3
        70 GOSUB 300
        80 STOP
        90 REM END OF MAIN PROGRAM

        200 REM SUBROUTINE TO COMPUTE NUMBER OF STARS TO PRINT
        210 B=I
        220 A=I-2
        230 GOSUB 300
        240 RETURN:REM RETURN TO MAIN PROGRAM

        300 REM SUBROUTINE PRINTS B-A STARS
        310 FOR J=1 TO B-A
        320 PRINT "*";
        330 NEXT J
        340 PRINT:REM PRINT A CARRIAGE RETURN
        350 RETURN
```

Notice that sometimes the subroutine at location 3ØØ is called by the subroutine at location 2ØØ, and it is called once by the main program. Whomever calls that subroutine is where the program returns on a RETURN statement.

Subroutines may be nested up to eight deep.

8.2 LINK Instruction

The LINK instruction allows you to transfer control from an NSC Tiny BASIC program to an INS8Ø73 assembly (machine) language subroutine.

Suppose Bill Counter has given you an assembly language subroutine that is perfect for counting widgets. You could convert Bill's program to NSC Tiny BASIC: but since you don't understand Bill's system for counting the widgets, and since Bill's system works, and since assembly language runs faster than NSC Tiny BASIC, etc., all you have to do is use the LINK instruction to transfer control from your NSC Tiny BASIC program to an assembly language subroutine.

A statement such as:

```
        >1Ø LINK #18ØØ
```

causes transfer control to the routine that starts at address location hexadecimal 18ØØ. There is a RET instruction at the end of the routine that returns you to your NSC Tiny BASIC program. RET is an assembly language instruction that acts like the NSC Tiny BASIC RETURN instruction: it returns you to the line number following Line 1Ø (the LINK statement).

Example:

```
>1Ø LINK #18ØØ  ◄──────────────────────────NSC Tiny BASIC trans-
>2Ø IF A=Ø THEN PR "SENSE A IS LOW"◄┬       fers to address #18ØØ
>3Ø IF A=1 THEN PR "SENSE A IS HIGH" ┤       to read sensor.
>99 STOP                             ┤
>RUN                                 └──Program transfers back
SENSE A IS HIGH                         to NSC Tiny BASIC

STOP AT 99
>RUN
SENSE A IS LOW

STOP AT 99

                .TITLE SENSE                   Assembly Language
  ØØØØ          .=Ø18ØØ     :HEXADECIMAL           "        "
  18ØØ 06       LD    A,S                          "        "
  18Ø1 D41Ø     AND   A,=16                        "        "
  18Ø3 6CØ2     BZ    LOW                          "        "
  18Ø5 C4Ø1     LD    A,=1                         "        "
  18Ø7 CAØØ     ST    A,Ø,P2                       "        "
  18Ø9 5C       RET◄—(This gets you back)          "        "
        ØØØØ    .END                               "        "
```

8.3 DELAY

Often, a program needs to give itself a pause to allow some external event to occur, or just to let you think for a moment.

For example, take the case where you have written a routine to ring the bell when your results are back from another program. If the program is a long one, you can go talk on the telephone until it is finished and you can hear the bell ringing. If the program is ended in a loop:

```
        980 PRINT "CNTRL G";:REM RING BELL
        990 GO TO 980:REM DO IT AGAIN
```

the terminal will continue to ring the bell, or the TTY will sound like an alarm clock, until you get away from the 'phone and stop the program.

A better way to handle this situation is to waste some time before the bell is rung again. Many programs do this with loops that waste time:

```
        980 PRINT "CTRL G";:REM RING BELL
        990 FOR I=1 TO 1000:NEXT I:REM WASTE TIME
        1000 GO TO 980:REM RING BELL AGAIN
```

unfortunately, this kind of time wasting is not precise and the number of times you go through the loop must be worked out by trial and error.

NSC Tiny BASIC has an inherently more precise method of generating time delays. This method is the use of the DELAY instruction, which can stop the processors operation for 1 to 1040 time units. If your INS 8073 is clocked by a precise 4 MHz timebase, these time units will be exact milliseconds. The example system in Section 3 of this manual uses such a crystal.

If you wanted your end-of-program bell to ring only once per second, as a gentle reminder for you to go to the system, all you need do is to change the initial example program:

```
        980 PRINT "CTRL G";:REM RING THE BELL
        985 DELAY 1000:REM WAIT ONE SECOND
        990 GO TO 980:REM DO IT AGAIN
```

Notice that the number (or expression) that follows a DELAY instruction is equal to the number of milliseconds required. If, however, you type "DELAY 0" the microinterpreter will default to the largest possible delay of 1040 milliseconds.

8.4 The ON Statement

Sometimes a program can't respond quickly enough to a stimulus through normal program operation. For example, take a system which must count widgets while calculating Pi to a million decimal places. The calculation of Pi will obviously take the "smartest" computer hours of calculations of Taylor series polynomials. The widgets are passing by on a conveyer belt at the rate of three per minute. Should the program to calculate Pi take a peek during every stage of its calculations just to look for a widget? The obvious answer is no as this would waste too much time: and the hours-long program might end up taking a week to execute.

The INTERRUPT can break into a program, perform some time intensive function, then allow normal operation to continue without any interference with the main program. An interrupt operates the same way you would if you were reading a book and the 'phone rang. First you'd save your place in the book, then you'd talk on the 'phone until your business was done, then you would hang up and go back to your book to the place where you left off.

The INS8073 handles interrupts with the ON statement. When you say:

```
        110 ON 1 250
```

you're saying, "If something (widget detector) puts a 0 on input SENSA/INTA on the INS8073, act like you first encountered a GOSUB 250".

There are two inputs on the INS8073, INTA and INTB. Correspondingly, there are two ON statements, ON1 and ON2. Unfortunately, the INTA input is also used for serial input for the RS-232 or TTY terminal. This means that the ON statement can't be used if you want to use a terminal with an NSC Tiny BASIC system. The ON instruction is, however, very useful in a ROM-based direct executing system.

An interrupt may be disabled at any time by executing the command:

```
        ON 1 0
```

which acts only on INTA. This can be used to put the microinterpreter into a "Don't bother me, I'm busy" state.

The following is a program that counts how much time has elapsed until an interrupt occurs, and, how many times it has been interrupted:

```
        10 REM TURN ON INTB
        20 ON 2 200
        30 A=A+1:REM HOW LONG SINCE LAST INTERRUPT?
        40 GO TO 30:REM KEEP COUNTING
        50 REM END OF MAIN PROGRAM
        200 REM START OF INTERRUPT ROUTINE
        210 REM A "GOSUB" TO THIS LOCATION IS GENERATED
        220 REM BY A HIGH TO LOW TRANSITION ON INTB
        230 B=A:REM STORE TIME BETWEEN INTERRUPTS
        240 C=C+1:REM COUNT HOW MANY INTERRUPTS HAVE OCCURRED.
        250 A=0:REM INITIALIZE THE COUNTER
        260 RETURN:REM KEEP WATCHING THE TIME
```

Although this program has no practical application, it should show you how to use the ON statement well enough to enlighten you about the basics of interrupts.

8.5 The STAT Function

There is a function in NSC Tiny BASIC which allows you to operate directly on the Status Register of the CPU. The status register can be loaded, or examined, through the use of the STAT function. This is another way of setting the interrupt enable bits on the processor, although it does not allow the assignment of a line number for the interrupt service routine. Therefore, the STAT function is not recommended for interrupt servicing.

The bits of the Status Register are defined as follows:

BIT NUMBER	FUNCTION	
7	CARRY	Not recommended for use in NSC Tiny BASIC
6	OVERFLOW	
5	SENSE A/INTA	May be examined as sense lines by the STAT operator
4	SENSE B/INTB	
3	FLAG 3	May be set using the STAT operator
2	FLAG 2	
1	FLAG 1	
Ø	INTERRUPT ENABLE	Not recommended for use with NSC Tiny BASIC

The SENSE A and SENSE B lines may be used as inputs and are read-only. If a serial terminal is being used to program the microinterpreter, SENSE A will already be occupied and SENSEB will have to be used.

The FLAG 1, 2 and 3 outputs are write-only, and FLAG 2 and FLAG 1 are used for the Read Relay and RS-232/TTY outputs respectively. Therefore, only FLAG 3 is available. You can see how they operate if you connect simple devices to one SENSE input and one FLAG output.

Assume that you have a source of slowly changing "1's" and "Ø's" coming into SENSE B. A simple switch would be a fine example. Also assume that an audio amplifier is attached to FLAG 3 so that you may hear the output. With the following program you can detect the switch position with your ears:

```
1Ø REM SENSE B TO FLAG 3 PROGRAM
2Ø A=STAT AND #1Ø:REM SENSE B BIT ONLY
3Ø IF A>Ø THEN GO TO 2Ø:REM SWITCH OPEN, NO SOUND
4Ø STAT=STAT OR #8:REM SET FLAG 3
5Ø DELAY 5:REM 1ØØ HERTS HALF WAVELENGTH
6Ø STAT=STAT AND #F7:REM CLEAR FLAG 3
7Ø DELAY 5:GO TO 2Ø:REM TEST SWITCH AGAIN
```

thus, you can use the STAT function to control minimal I/O in your system.

If you insist on using STAT to set the Interrupt Enable bit, be aware that that bit will not be set until after the end of the next instruction. This gives you some time to prepare.

8.6 Multiprocessing, INC (X), DEC (X)

The INC and DEC, or Increment memory location and Decrement memory location instructions are provided to facilitate using NSC Tiny BASIC in a multiprocessor environment.

Multiprocessing is an art in itself and is considerably beyond the scope of this primer. If you require more information on multiprocessing, refer to the "70 Series Microprocessors User's Manual".

If you are familiar with the techniques of multiprocessing with the INS8070 series of microprocessors, then you are familiar with the attributes of the ILD (Increment and Load) and DLD (Decrement and Load) instructions. INC and DEC provide the same function in NSC Tiny BASIC format. The instructions are non-interruptable and can be used for semaphores between microprocessors.

If you choose to use the INS8073 in a bus-coupled microprocessor system one precautionary note is given: the variable RAM at location X'1000-X'10FF must be separate for each processor on the bus. If this is not observed, FOR loops, subroutines, and even the variables A through Z will become hopelessly garbled. All other external memory may be shared.

8.7 CLEAR

The CLEAR statement is used to zero all variables, and terminate all pending interrupts and loops. This statement should be used with extreme caution as it can terminate program execution. When properly used, it can be a boon in setting up initial conditions within a program.

CHAPTER 9

9.1 Memory Organization

In order to use NSC Tiny BASIC, you must have an INS8073 system with a minimum of 256 read/write memory locations needed to store the variables from A to Z and to accomplish other housekeeping functions. In most cases more memory is needed: a typical system should have at least 2K (2048) bytes of RAM. Each memory location stores one byte, and in a typical system with 2K (2048) bytes of RAM, each location has a unique memory address running from 4096 to 6143. The microinterpreter will only see RAM locations that are contiguous (non-stop with RAM starting at the location 4096.

The memory is organized as follows:

1. The first 2560 locations consist of NSC Tiny BASIC, in on-chip ROM (Read Only Memory) on the INS8070 chip. In other words, NSC Tiny BASIC consists of 2560 bytes of pre-programmed memory occupying locations 0 to 2559.

2. The next 1536 locations are unassigned and can be used for ROM, data RAM or I/O devices.

3. The next 256 or more locations, with addresses from 4096 to 65470 if desired consist of RAM (Random Access Memory, also called Read/Write Memory). This part of memory serves two purposes:

   a. Locations 4096 to 4351 are used by NSC Tiny BASIC as a "scratchpad memory". They are not available for your use.

   b. Locations 4352 to your last RAM location are yours to use. When you type:

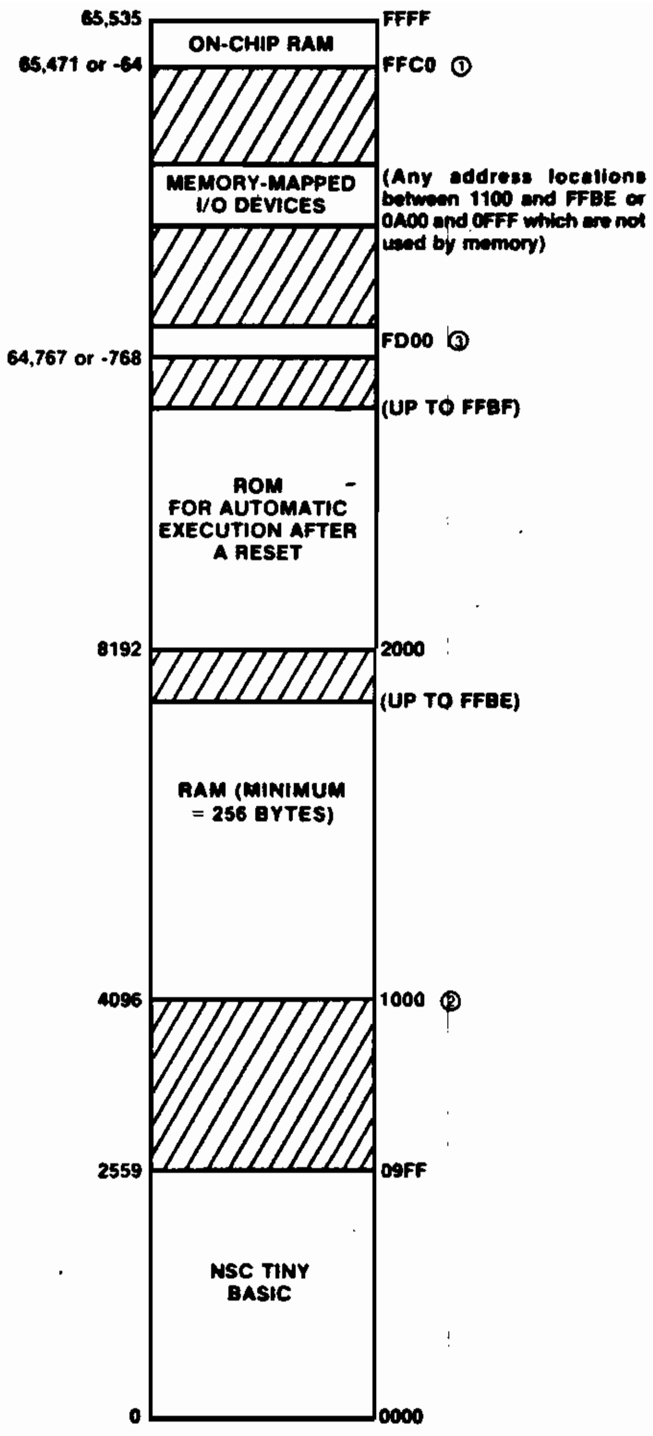
      ```
      NEW #1000
      NEW
      ```

      then store a program (with line numbers), your program is stored in memory, beginning at location 4352.

4. If automatic ROM execution is desired after a RESET, the ROM program must start at location 8192, and can extend up to 65471.

5. I/O devices may be memory-mapped in any unused memory locations in the ranges 2560-4095 and 4352-65470.

6. No RAM or I/O device can occupy memory location 65471. I/O devices should not be mapped contiguous with RAM from 4096.


65,535
FFFF
ON-CHIP RAM
65,471 or -64
FFC0 ①
MEMORY-MAPPED I/O DEVICES
(Any address locations between 1100 and FFBE or 0A00 and 0FFF which are not used by memory)
FD00 ③
64,767 or -768
(UP TO FFBF)
ROM FOR AUTOMATIC EXECUTION AFTER A RESET
8192
2000
(UP TO FFBE)
RAM (MINIMUM = 256 BYTES)
4096
1000 ②
2559
09FF
NSC TINY BASIC
0
0000

NOTE 1. RAM or I/O devices must not occupy location X'FFBF.

NOTE 2. The microinterpreter will assume the only available RAM is that which starts at location X'1000 and ends at the first discontinuity encountered above that address.

NOTE 3. Location X'FD00 must be used to set the baud rate of the console device. If no console is used, this location may be used as desired.

Figure 9-1. NIBL Memory Diagram

The following table summarizes the memory organization in a minimum NSC Tiny BASIC system.

Table NSC Tiny BASIC Memory Organization

LOCATIONS	CONTENTS
0 - 2559	NSC Tiny BASIC System (ROM)
4096 - 4351	Scratchpad Memory, (RAM)
4352 - 6143	User Space. Your programs are stored here.

User space, locations 4352-6143 will hold 1792 bytes: this is enough to store approximately sixty NSC Tiny BASIC statements. Additional memory can be added: and, if your system has 4096 bytes of RAM, the user space locations run from 4352 to 8191.

The memory layout for the example board is given in Section 3.

9.2 TOP Location

The first location that is free for your use has a special name, it's called TOP. The statement:

```
        >PRINT TOP
```

will cause the address of the first free location to be printed.

To see how TOP works, clear away any old program by typing NEW #1000 then NEW, print out the value of TOP, then store a line or two of any program and try printing TOP again.

```
        >NEW #1000
        >NEW
        >PRINT TOP
         4353 <------Remember 4352 is the beginning of "user space".
                     A program (even with no lines) takes up one byte.

        >10 REM THIS TAKES UP SPACE

        >PRINT TOP
         4380 <------Locations 4352 through 4379 are in "use".  First
                     available is now 4380.
```

A longer program will use up more space.

```
        >NEW #1000
        >NEW
        >PRINT TOP
         4353 <------TOP points to the beginning of NSC Tiny BASIC's
                     user space.
```

```
>100 REMARK POWERS OF TWO
>110 P=1
>120 PRINT P;
>130 P=2*P
>140 IF P=64 GO TO 120
>999 STOP

>PRINT TOP
 4440 <-------Next available location is now 4440.
```

Remember, the value of TOP is the address of the next available memory location beyond the last byte of your NSC Tiny BASIC program.

Choose a safe location far away from the small program that you will write shortly, for example location 5000. You want to store a number into location 5000, that is, you want to put a number (say 55) at 5000.

```
>@5000=55
```

The @ is the familiar "at" symbol and means "at the location". Remember, location 5000 is an actual memory location and not an output port. If you could peek into location 5000 you'd now find the number 55 resting there; however, since you cannot "see into" locations, tell NSC Tiny BASIC to print a copy of whatever is stored there.

```
>PRINT @5000
 55
```

Try some more.

```
>@5001=37 <-------Put 37 into location 5001

>PRINT @5001
 37

>@5002=@5000+@5001

>PRINT @5002
 92
```

Did you follow the last example? You previously had put 55 into location 5000 and 37 into 5001, you can add them (@5000+@5001) and put the result into 5002.

You can use @5000, @5001, @5002 and so on just as you use variables A through Z, except for one thing:

> Numerically addressed locations can store one byte only. They accept numerical values from 0 to 255, inclusive. You cannot store negative numbers or numbers larger than 255 in these locations.

As you may suspect, the variables A through Z each occupy two bytes in the INS8073's memory.

The following illustrates what would happen if you tried to put a number larger than 255 into a numerically addressed location:

```
>@5004=256

>PRINT @5004
 0

>@5005=257              Try some negative numbers
                        and see the results.

>PRINT @5005
 1

>@5006=511

>PRINT @5006
 255
```

Attempting to put a number into a memory location that is too large or too small for that memory location will not result in an error message. The number will be treated modulo 256 (that is, it will be divided by 256 and the remainder put into the location).

One trick to using memory locations is to call them TOP+1, TOP+2 and so on: as your program changes size, a notation such as:

```
>@(TOP+23)=211
```

is always above your program. A check for the top of memory can be done easily using the IF statement (assuming that M is the number of the memory location you were about to use, and that your memory went up to location 5143):

```
>70 IF M>5143 THEN PRINT "OUT OF MEMORY"
```

In summary: to put a value "V" into a memory location M write

```
>@M=V
```

and the value of the memory location M is given by the expression @M.

The at sign (@) should be used with caution. Placing a value in memory used by your program (at a location less than TOP) can cause the program to "blow up". This means that it refuses to work, and there may be no way to LIST or otherwise preserve it. Even if it doesn't blow up other insidious changes that can be hard to find can occur. Be careful.

The following lists the locations that should not be used:

LOCATIONS	
0 to 2559	ROM (on-chip)
-64 to -1	RAM (on-chip)
4096 to (TOP-1)	program in our typical system

9.3 Strings of Characters

An important feature of NSC Tiny BASIC is its ability to input, output and manipulate characters as well as numbers. As you have already seen, the statement:

```
>220 PRINT "FLOW OK"
```

will cause NSC Tiny BASIC to print the words:

```
FLOW OK
```

The information between quotes is called a string.

The computer can store strings, recall them and do other operations on them as well. These abilities are especially handy where the user of a program should communicate in something resembling natural language. It might be more convenient to have a user type YES as an answer to a question rather than have the computer type "ENTER 1 IF YOU MEAN YES OR 0 IF YOU MEAN NO".

To have a user of your program utilize string input, you have to first decide where the computer will put the string. The previously described TOP function gives the first location in memory that is available to the user. If the program assigns:

```
S=TOP
```

then S will be the address of the first location in memory that can be used. In the following case it will be used for storing a string. When you want to work with strings instead of numbers, use the dollar sign ($) to tell NSC Tiny BASIC to expect a string. The statement:

```
INPUT $S
```

stores whatever string the user types beginning at location S. The first character of the string goes right at location S, the next character at location S+1 and so on. Input stops when the RETURN key is pressed. The code for the RETURN key is stored at the last character of the string. This is important as it allows you to find the end of the string later.

Try the following program, note the $ sign in Lines 20 and 30.

```
>NEW

>10 S=TOP+100:REM SET S TO POINT AT A FREE SPOT IN MEMORY

>20 INPUT $S:REM GET SOME CHARACTERS

>30 PRINT $S:REM TYPE THE CHARACTERS JUST OBTAINED

>RUN
?ABC ◄──────ABC is the input string.
ABC

>RUN
? SAM 123 ◄──────This is a California license plate number.
SAM 123

>RUN
?%@7#Q+*! ◄──────You can use just about any TTY character in
%@7#Q+!          a string.

>                And so on.  Try some of your own.
```

In the second RUN above, we typed SAM 123 and pressed RETURN. Therefore, a string of eight characters will be stored, beginning at TOP+100, as follows:

LOCATION	CONTENTS
TOP+100	S
TOP+101	A
TOP+102	M
TOP+103	space
TOP+104	1
TOP+105	2
TOP+106	3
TOP+107	RETURN key code

Remember, each location stores one byte, so each character or key code is stored as one byte code.

In the following program, we use a string variable $R:

```
>NEW

>60 R=TOP+10:REM SET R TO POINT AT FREE SPOT IN MEMORY
>70 $R="ABCDEFGHIJKLMNOPQRSTUVWXYZ"
>80 PRINT $R
>RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

In the preceding program, $R is a string variable. In Line 70, you assign it a string value consisting of the 26 letters of the alphabet. Note that these letters are enclosed in quotation marks; however, the quotation marks are not stored as part of $R. The string in the $R will be stored in TOP+10 through TOP+35 and a RETURN key code will be put into TOP+36 to mark the end of the string.

Add the following lines to the above program, (don't type NEW!)

```
        >90 $R="HELLO"
        >100 PRINT $R
        >RUN
        ABCDEFGHIJKLMNOPQRSTUVWXYZ
        HELLO
```

It is your responsibility, as a programmer, to see that there is enough space for strings. For example, add the following lines to the program we are developing:

```
        >110 E=TOP+12
        >120 $E="123"
        >130 PRINT $R:REM YES WE DO MEAN $R AND NOT $E
        >RUN
         ABC.....
         HELLO
         E123
```

Note the strange result when this part of the program is RUN. This demonstrates that you have to be able to guess about how long strings are going to be when you decide where to put them in memory.

Try other strings for $R and $E in this program, and see what conditions cause overlap, and by how much. Change the constant in Line 110 as well. A few experiments will teach more than a thousand words of text.

String characters are actually stored as numbers. There is a standard numerical code for each TTY character, called ASCII (American Standard Code for Information Interchange). This code is used by all manufacturers of computers and communication equipment. There are other codes in use too, but only by a small number of manufacturers, and they make ASCII available to their equipment. It is easy to write a program that will show the ASCII code that NSC Tiny BASIC uses to store string characters. This program will print the ASCII code for a character as a decimal number after you have typed in the character and hit the RETURN key.

```
        >LIST
        100 REM PROGRAM TO PRINT ASCII CODES FOR TTY CHARACTERS
        110 A=TOP+500:REM LOCATION TO PUT CHARACTERS
        120 PRINT "EACH TIME I TYPE A QUESTION MARK, YOU TYPE"
        130 PRINT "A SINGLE CHARACTER AND HIT THE RETURN KEY"
        140 PRINT "":INPUT $A:REM GET A CHARACTER
        150 PRINT @A:REM PRINT ASCII CODE
        160 GO TO 140:REM DO IT AGAIN
```

```
>RUN
EACH TIME I TYPE A QUESTION MARK, YOU TYPE A
SINGLE CHARACTER AND HIT THE RETURN KEY

?A
 65

?B
 66

?C
 67

?  <——————This is CONTROL G (used ring the BELL).  It is a
 7         non-printing character.

?%
 37

?0
 48

?1
 49

?2
 50

? ... Your turn, experiment.
```

Using this program, look at the ASCII codes of the letters, numerals, and special characters on the TTY keyboard. Remember, NSC Tiny BASIC stores the RETURN character at the end of each string. How would you print out the ASCII code for the RETURN key? (See Appendix C)

Using the program as written, find the ASCII codes for CONTROL A, CONTROL B and so on. There is a problem trying to print the code for CONTROL C. Can you deduce its value? (See Appendix C)

The last string feature in NSC Tiny BASIC is string replacement. If P and Q are suitably defined (as pointing to memory) then a statement such as:

```
>500 $P=$Q
```

will take the string starting at location Q and make a copy string starting at location P. Remember, it is up to the programmer to be sure that there is enough room for this to occur. A real disaster can occur if P=Q+1. For instance, when Line 500 is executed, the character at location Q is placed in location P. But, location P is the second location of Q! (Remember P=Q+1.) This means that the first character of Q is now also the second character of Q. Since this is a string copy instruction the next thing that happens is that the second character of Q is copied into the second character of P. The second character of Q was just copied from the first character Q, so the second character of P is the same as the first character of Q. Now, since P=Q+1 the second character of P is the same as the third character of Q. And so it goes, with the first character of Q being copied over and over again. The process will never stop: if there was a RETURN somewhere in P it will be "clobbered" by the constantly copied character. Soon all memory will be filled by this one character, your program will be destroyed, and, NSC Tiny BASIC will come to a grinding halt. Be careful to avoid round robin situations like this one. Try it once.

To compare strings in an IF statement you must compare the ASCII values since NSC Tiny BASIC doesn't allow direct comparison of strings. This merely means using @ instead of $, and doing the comparison one memory location at a time.

9.4 Exercise

Write a program to compare two INPUT strings and print "THE STRINGS ARE EQUAL" if they are, and "THE STRINGS ARE UNEQUAL" if that is the case. The following is a part of the program:

```
>LIST

100 REM PROGRAM TO COMPARE TWO STRINGS
110 PRINT "THIS PROGRAM COMPARES TWO STRINGS AND TELLS"
120 PRINT "YOU WHETHER THEY ARE EQUAL OR UNEQUAL."
130 A=TOP+100
140 B=TOP+200
150 PRINT "":PRINT "FIRST STRING";:INPUT $A
160 PRINT "SECOND STRING";:INPUT $B
170 GOSUB 1010:REM GO COMPARE STRINGS
180 GO TO 150:REM GET TWO MORE STRINGS
1000 REM SUBROUTINE TO COMPARE STRINGS AND PRINT MESSAGE
1010 Your work begins here...
```

Write the subroutine to compare the strings and print the appropriate message. A RUN of the complete program might look like this:

```
>RUN
THIS PROGRAM COMPARES TWO STRINGS AND TELL YOU WHETHER THEY
ARE EQUAL OR UNEQUAL.

FIRST STRING? ABC
SECOND STRING? ABC
THE STRINGS ARE EQUAL

FIRST STRING? ABC
SECOND STRING? DEF
THE STRINGS ARE UNEQUAL
```

```
FIRST STRING? AB
SECOND STRING? ABC
THE STRINGS ARE UNEQUAL

FIRST STRING? ABCD
SECOND STRING? ABC
THE STRINGS ARE UNEQUAL

FIRST STRING? A BC -------The space is a part of the string
SECOND STRING? ABC
THE STRINGS ARE UNEQUAL

FIRST STRING?
```

See Appendix A for answers

CHAPTER 10

10.1 Interfacing Other Devices To NSC Tiny BASIC

Devices other than the TTY or terminal can be attached to the example system via the memory bus. The INS8073 Data Sheet contains the pin assignments and interfacing data needed to talk to it via the bus. This chapter describes a simple circuit for you to wire up and plug into the INS8073. Then you will be guided in writing several simple control programs to exercise the circuit. Our circuit is very simple, consisting of a switch and a LED, with the INS8073 in between.

$$\begin{tikzcd}
\boxed{\text{SWITCH}} \arrow[r] & \boxed{\text{INS8073}} \arrow[r] & \boxed{\text{LED}}
\end{tikzcd}$$

An I/O device looks like a memory location both in hardware and in software. It decodes an address and accepts or sends a byte of data.

For this chapter, the reader is assumed to be familiar with digital logic, the various forms of binary and hexadecimal notation and the other mental equipment usually acquired by those who design digital electronic circuits.

In NSC Tiny BASIC the 16 bit address corresponds to the signed numbers from -32768 to 32767. The high-order bit, instead of being treated as a sign bit, becomes simply the high order bit of the address. The simplest way to address locations above 32767 is to use the hexadecimal format. 32767 = #7FFF. Negative decimal numbers in NSC Tiny BASIC where the high order bit is the sign bit, are twos complement 16 bit binary representations. Thus -1 (in binary 1111 1111 1111 1111) used as an address would access the same memory location as #FFFF. All in all, it is clearer to use hexadecimal notation in NSC Tiny BASIC for addressing high memory locations.

The timing considerations for address and data set up and strobes can be found in the INS8073 Data Sheet. Usually the NSC Tiny BASIC program itself does not have to be concerned about output timing as NSC Tiny BASIC is very slow with respect to TTL or any other semiconductor technology. Almost any circuit can easily follow the output from an NSC Tiny BASIC program. On the other hand, it is easy to feed data to the computer too quckly for NSC Tiny BASIC to follow. For many control applications, a response time on the order of a second is adequate, and in those cases NSC Tiny BASIC can be used in an on-line device. Faster response can be obtained by using interrupts or programming in the INS8073 assembly language. National Semiconductor Publication Number uPG-420306255-001 describes the facilities of the assembler.

Even if the assembler is to be used, NSC Tiny BASIC is still a good way to check out the algorithms and the interfaces quickly and inexpensively. Use of the assembler is considerably more time consuming and costly than writing in NSC Tiny BASIC.

10.2 Hardware Interface

The circuit, shown below, is in two parts. The first part lights an LED when the appropriate NSC Tiny BASIC command is given. Instead of an LED the circuit could have a relay, or other device that is to be controlled. The LED, of course, could also be part of an opto-isolator or the input to a solid state relay.

The NSC Tiny BASIC statement:

```
        >@#7FFF=1
```

puts the value 1 at location (in hexadecimal) #7FFF. This location, instead of being a memory location, is used for I/O. The hardware you are constructing has to recognize when it is being addressed. This occurs when the number #7FFF appears on the address lines, labelled A0 through A15 (pins 9 through 19 and 21 through 25 on the INS8073 As it happens #7FFF in binary is 0111 1111 1111 1111 so you want to recognize when all address lines are high. There are a number of ways to do this. Three DM8131 bus comparators would do, but you may choose the more elementary method of ANDing the lines together. To do this two DM74LS30 eight input NAND gates are used. The output of the DM74LS30 is low only when all eight inputs are high. Thus the address is correct for the device when the outputs from both DM74LS30'S are low.

The two outputs from the DM74LS30'S go into one NOR gate of the quad NOR (a DM74LS02). This NOR is high when both inputs are low. You will need a low when the outputs from the DM74LS30'S are low: therefore, ADM74LS04 is used as an inverter.

Now the circuit can detect its address, but all kinds of signals appear on the address bus when a program is running. Therefore, another line NWDS (pin 6) is on the bus. This line is normally high but goes low when the CPU puts an address on the bus as part of a memory write instruction. This is the only time that you want the circuit to "look at" the address lines. Another section of the DM74LS02 detects when the address is #7FFF at the same time NWDS is low. At such times the output of this NOR gate goes high. This signal clocks one of the flip-flops of a DM74LS74. This is a D type flip-flop so that when the clock makes an upward transition the logic level at the data input is copied to the Q output: it is held at that level until the next positive edge on the clock triggers the flip-flop. Thus the DM74LS74 captures the data from the bus on data line D0 (pin 33). Any of the data bits could have been used, this is an arbitrary choice: in fact, by using four DM74LS74's all eight bits could be used.

This half of the circuitry can be summarized as follows: when the proper address appears on the address bus, and NWDS is active, the low order data bit appears on the output, and is held there. This bit is used to light an LED.

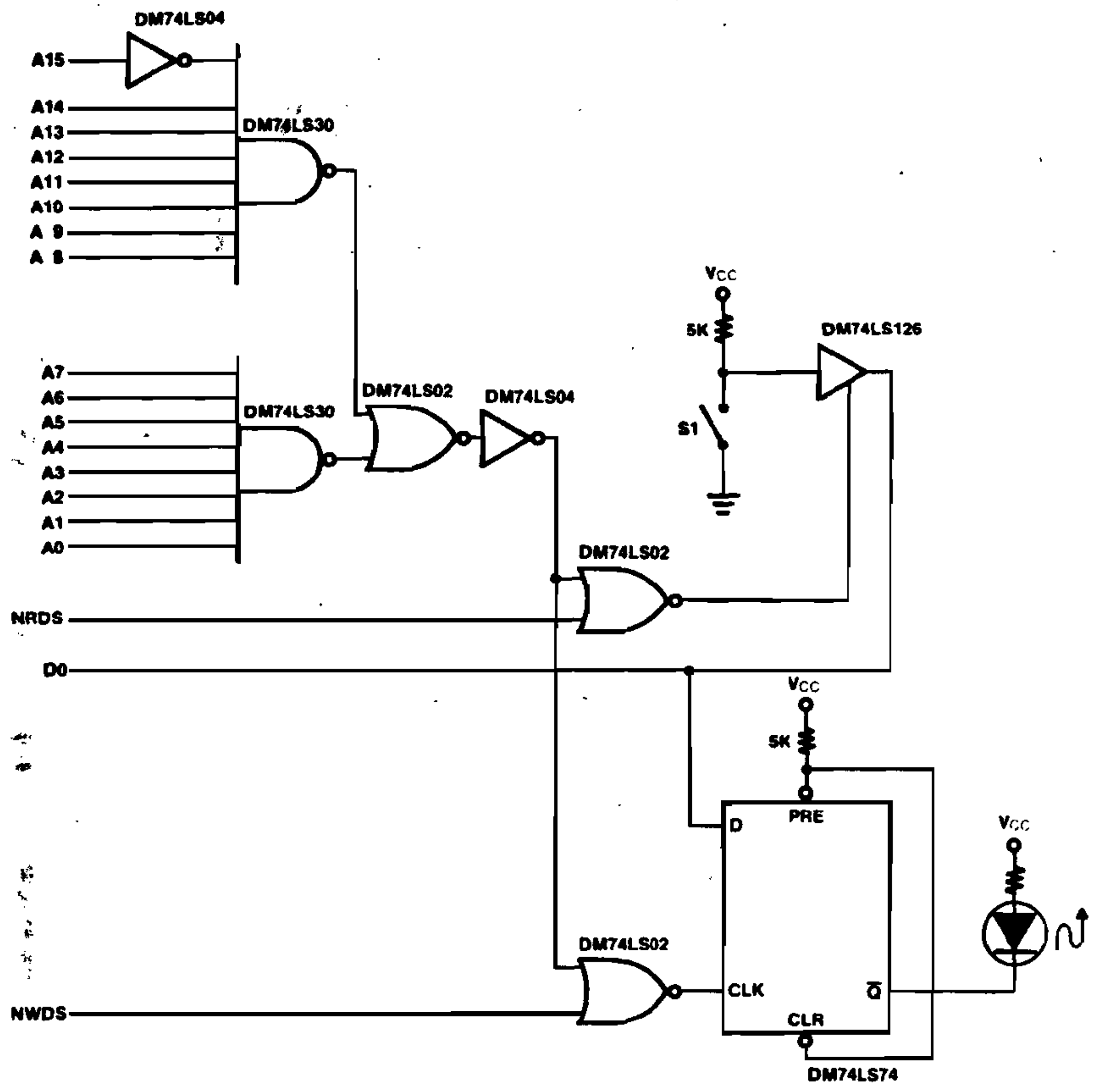


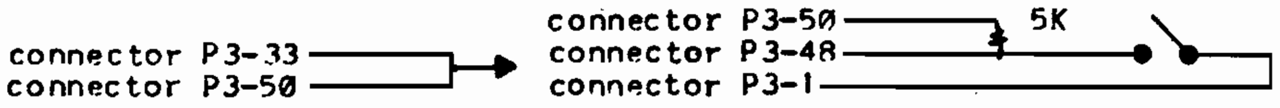
Figure 10-1. LED I/O Schematic Diagram

Inout to a computer is simpler. A pullup resistor and a switch puts a logic Ø (switch closed) or a logic 1 (switch open) on the input of a TRI-STATE buffer (DM74LS126). This is the desired logic level we wish to communicate to the computer. The output of the buffer is fed to the same bit Ø of the data bus. This demonstrates the bi-directional nature of the bus. The same pin (pin 33 which was just used for output is now used for input. The computer knows which is which by putting another signal NRDS (pin 4) on the bus whenever it wants data.

When NRDS is low the computer expects the circuit you are building to place data on the data lines. The TRI-STATE buffer is in its high-impedance state, thus not affecting the bus, until a signal gates it. The signal is the NOR of NRDS and the address circuit already described. Thus the value of the switch is put on the bus only when NRDS is active and the correct address is on the address bus. Another method is available to the user of the example system shown in Section 3.

10.3 Example System LED Flasher

An easy way to attach a switch and LED to the example board is shown below. The LED is connected to the output of the 8154's I/O port, and a switch is connected to a different port.



Before any programs are operated, the INS8154, which controls these pins, must be initialized. Reasoning behind this can be found in the Data Sheet for that device. To set the INS8154 to output to the LED, type:

```
>@#9AA2=Ø1
```

Changes will also have to be made in the preceding programs to reflect that:

1. The switch is now read as bit Ø of address #9AA1
2. The LED is now bit Ø of location #9AA1
3. The LED now lights when given a Ø input and goes dark when its bit is set to a 1.

Once these few changes have been made, the LED flasher problems can be implemented for the circuit the same as the other.

There are many other ways of implementing these functions, and this manual is not intended to instruct in hardware design. This circuit is presented as material for a programming exercise only.

Placing an inverter between any of the address lines and the inputs to the DM7430'S will require that bit to be zero in order to address this device. Thus any address can be used if #EFFF is not appropriate for your system.

10.4 Programming the Circuit

It is assumed that you now have the circuit wired up, ready to test. To test the circuit, type the following NSC Tiny BASIC statements and watch the results.

```
        Turn the LED ON

        >@#7FFF=1    -------The LED should come ON

        Turn the LED OFF

        >@#7FFF=0    -------The LED should go OFF

        Open the switch and type

        >PRINT @#7FFF
         1    -----------If the switch is open, you should get 1

        Close the switch and type

        >PRINT @#7FFF
         0    -----------If the switch is closed, you should get 0
```

If the above didn't happen, double check your I/O circuit before proceeding.

The following program senses the position of the switch and makes the light behave accordingly:

```
        >100 M=#7FFF:REM PUT THE DEVICE ADDRESS IN M

        >110 S=@M:REM SAVE THE VALUE OF THE SWITCH IN S

        >130 @M=S:REM SEND THE VALUE OF THE SWITCH TO THE LIGHT

        >140 GO TO 110:REM REPEAT, KEEP CHECKING SWITCH
```

This program could be shortened to:

```
        >50 @#7FFF=@#7FFF

        >60 GO TO 50
```

but it is not as clear that way. Going back to the first program, you can see one of the advantages of software over hardware. If you want to change the sense of the switch, have it on when it used to be off and vice versa, all you need to do is change Line 130 to:

```
        >130 @M=NOT (S)
```

and the switch works the other way around, without changing a single wire. Now the light is ON when the switch is closed and OFF when the switch is open. While it is not hard to change a wire, if this were part of a device committed to a printed circuit board, it might be quite expensive to either modify all the boards or have a new design put into production. The software change is often far simpler.

Suppose you want the light to be OFF when the switch is closed and blink ON and OFF when the switch is open.

```
100 M=#7FFF:REM PUT THE DEVICE ADDRESS IN M
110 S=@M:REM SAVE THE VALUE OF THE SWITCH IN S
130 REM IF SWITCH OPEN, BLINK LIGHT ON AND OFF
140 @M=S:GOSUB 210:REM LIGHT FOLLOWS SWITCH ON AND DO A TIME DELAY
150 @M=0:GOSUB 210:REM TURN LIGHT OFF AND DO A TIME DELAY
160 GO TO 110
200 REM TIME DELAY SUBROUTINE
210 T=100:REM MAKE T BIGGER TO INCREASE DELAY
220 DELAY T
230 RETURN
```

10.5 Exercises

Rewrite the above program so that the light blinks when the switch is closed and the light is OFF when the switch is open.

Write a program so that the switch must be closed for several seconds before the light comes ON. If the light is ON, opening the switch turns it OFF immediately. However, if the light is OFF and the switch is closed, several seconds must elapse before the light comes ON. If the switch is opened during this time, the light will not come on, or even blink.

Answers are in Appendix A

Section 2

CHAPTER 1

1.1 Introduction

This reference guide is intended to provide you with information on the use of NSC Tiny BASIC language. This section will also provide you with information on NSC Tiny BASIC commands, statements, grammar, error messages, and control characters. A brief description of each is given along with a short example or two to demonstrate their use.

This reference guide will provide a quick method of locating basic information on NSC Tiny BASIC. For a more detailed description, and examples of NSC Tiny BASIC's use, Section 1 should be consulted.

To learn how to use NSC Tiny BASIC, you will need an INS8073 system and a teletype or CRT terminal.

CHAPTER 2

2.1 Language Expressions

2.1.1 Variables

There are twenty-six variable names which can be used with NSC Tiny BASIC. These are the letters of the English language alphabet, A through Z. The values assigned to these variables are 16-bit signed integers. There are no fractions or floating point numbers.

2.1.2 Constants

All numeric constants are decimal numbers except when preceded by a pound sign (#). If preceded by #, the number is interpreted as a hexadecimal number. The symbols 55 would be treated as a decimal number, while #55 would be treated as a hexadecimal number (equal to 85 in decimal value). Decimal constants may be in the range of -32767 to 32767.

2.1.3 Relational Operators

Relational Operators are the standard BASIC symbols:

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

The relational operators return either a Ø (FALSE) or -1 (TRUE) as a result. NOTE: >< is an illegal operator.

2.1.4 Arithmetic Operators

Standard arithmetic operators are provided for the four basic arithmetic functions.

+	addition
-	subtraction
/	division
*	multiplication

Arithmetic is accomplished by standard 16-bit twos-compliment arithmetic. Fractional quotients are truncated, not rounded; therefore, 16/3 will give 5, 17/3 will also give 5 as a result. Remainders resulting from division are dropped. No attempt is made to round off the quotient. As usual, division by zero is not permitted; it will result in an error break.

The usual algebraic rules for order in evaluating expressions is followed. The order of evaluation is controlled by parentheses, and their liberal use is advised. They provide clarity and avoid confusion in complicated expressions.

2.1.5 Logical Operators

NSC Tiny BASIC provides Logical Operators AND, OR and NOT in addition to the arithmetic operators. These perform bitwise logical operations on their 16-bit arguments and produce 16-bit results. The AND and OR operators are called binary operators because they perform an operation on TWO arguments (or operands). An example follows with binary interpretation:

```
>LIST
 10 A = 75                      A = 0000 0000 0100 1011
 20 B = 99                      B = 0000 0000 0110 0011
 30 C = A AND B                 C = 0000 0000 0100 0011
 40 PRINT C

>RUN
67
```

2.1.6 Logical AND

```
>LIST
10 INPUT A
20 INPUT B
30 IF (A>50) AND (B>50) THEN GO TO 60
40 PRINT "ONE OR BOTH ARE SMALL"
50 GO TO 10
60 PRINT "BOTH ARE BIG"
70 GO TO 10

>RUN
? 51
? 52
BOTH ARE BIG
? 51
? 49
ONE OR BOTH ARE SMALL
? 49
? 49
ONE OR BOTH ARE SMALL
?^C
STOP AT 10
>
```

2.1.7 Logical OR

```
>LIST
1Ø INPUT A
2Ø INPUT B
3Ø IF (A>5Ø) OR (B>5Ø) THEN GO TO 6Ø
4Ø PRINT "BOTH ARE SMALL"
5Ø GO TO 1Ø
6Ø PRINT "ONE OR BOTH ARE BIG"
7Ø GO TO 1Ø

>RUN
? 51
? 52
ONE OR BOTH ARE BIG
? 51
? 49
ONE OR BOTH ARE BIG
? 49
? 49
BOTH ARE SMALL
?^C
STOP AT 1Ø
>
```

2.1.8 Logical NOT

The third logical operator (NOT) is a unary operator. It performs an operation on only ONE argument, as follows:

```
>LIST
>1Ø A = 11                  A = ØØØØ ØØØØ ØØØØ 1Ø11 = 11
                                                      1Ø
>2Ø B = NOT A
>3Ø PRINT B                 B = 1111 1111 1111 Ø1ØØ = -12
>RUN                                                   1Ø
-12
```

2.2 Functions

There are several functions that may be used in arithmetic expressions in NSC Tiny BASIC. These are described below.

2.2.1 MOD (a,b) Function

Returns the absolute value of the remainder a/b, where a and b are arbitrary expressions. If the value of b is zero, an error break will occur as in any division operation. As an example:

```
>1Ø A = 95                      2
>2Ø B = 44                 44/ 95
>3Ø PRINT MOD (A,B)            88
>RUN                            7  ------MOD (95,44)
 7
```

2.2.2 RND (a,b) Function

Returns a pseudo-random integer in the range of a through b, inclusive. For the function to perform correctly, a, should be less than, b, and b-a must be less than or equal to 32767 (base 10). A typical example is:

```
>10 PRINT RND (1,100)
>RUN
 27
```

2.2.3 STAT Function

Returns the 8-bit value of the INS8073 Status Register. STAT may appear on both sides of an Assignment Statement: so, the programmer can modify the Status Register as well as read it. The Carry and Overflow Flags of the register are usually meaningless, since the NSC Tiny BASIC interpreter itself is continually modifying these flags. The Interrupt-Enable Flag may be altered by an assignment to STAT these flags. The Interrupt-Enable Flag may be altered by an assignment to STAT (such as: STAT = #FF). Location of individual flags are shown below:

Most Significant Bit — Least Significant Bit

7	6	5	4	3	2	1	0
CY/L	OV	SB	SA	F3	F2	F1	IE

Example of use:

```
>10 LET A = STAT
>20 PRINT A
>RUN
 176  --------The decimal number, 176, translates to:
              1 0 1 1 0 0 0 0 binary.
```

2.2.4 Status Register Bit Functions

The function of each bit in the Status Register is described below:

BIT	DESCRIPTION
7	CARRY/LINK (CY/L): This bit is set to 1 if a carry occurs from the most significant bit during an add, a compliment-and-add, or a decimal-add machine language instruction. This bit may also be set by the operations performed by the SHIFT RIGHT WITH LINK (SRL) and the ROTATE RIGHT WITH LINK (RRL) machine language instructions. CY/L is input as a carry into the bit 0 position of the add, compliment-and-add, and decimal-add machine language instructions.

6 OVERFLOW (OV): This bit is set if an arithmetic overflow occurs during an add (ADD, ADI or ADE) or compliment-and-add (CAD, CAI or CAE) machine language instructions. Overflow is not affected by decimal-add (DAD, DAI or DAE) machine language instructions.

NOTE: The above two bits may be of little or no use in an NSC Tiny BASIC program.

5 SENSE BIT B (SB): This bit is tied to an external connector pin and may be used to sense external conditions. This is a "read-only" bit; therefore, it is not affected when the contents of the accumulator are copied into the status register by a STAT instruction. It is also the second interrupt input and may be examined by the "ON" command.

4 SENSE BIT A (SA): This bit is also tied to an external connector pin. It serves, as does SENSE BIT B, to sense external conditions. In addition, it acts as the interrupt input when the INTERRUPT ENABLE (see bit 3 of status register) is set. This bit is also a "read-only" bit. The same "ON" command may be used to sense this input. This flag is used by NSC Tiny BASIC as the serial input bit from the TTY or CRT.

3 USER FLAG 3 (F3): This bit can be set or reset as a control function for external events or for software status. It is available as an external output from the INS8Ø73.

2 USER FLAG 2 (F2): Same as F3. This flag is used by NSC Tiny BASIC to control the paper tape reader relay.

1 USER FLAG 1 (F1): Same as F3. This flag is used by NSC Tiny BASIC as the serial output bit (with inverted data) to the TTY or CRT.

NOTE: The flag 1, 2 and 3 outputs of the status register serve as latched flags. They are set to the specified state when the contents of the accumulator are copied into the status register. They remain in that state until the contents of the status register are modified under program control.

Ø INTERRUPT ENABLE FLAG (IE): The processor recognizes the interrupt inputs if this flag is set. This bit can be set and reset under program control. When set, NSC Tiny BASIC recognizes external interrupt requests received via the SENSE A or B inputs. When reset, it inhibits the INS8Ø73 from recognizing interrupt requests.

2.2.5 TOP Function

Returns the address of the first byte above the NSC Tiny BASIC program in the current page which is available to the user. This will be the address of the highest byte in the NSC Tiny BASIC program plus 1. All the memory in the RAM above and including TOP can be used by the NSC Tiny BASIC program as scratchpad storage. As an example:

```
        >1Ø PRINT TOP
        >RUN
        44ØØ     ------44ØØ is the first address of unused RAM
```

2.2.6 INC (X) and DEC (X) Functions

These statements increment or decrement a memory location X. Examples:

```
        >1Ø LET X=1Ø32
        >2Ø INC (X)
             .
             .
             .
             .
        >5Ø DEC (X)
        >6Ø INC (6ØØØ)
        >7Ø DEC (6ØØ1)
```

These instructions are used for multiprocessing and are non-interruptable. This means that if two 8Ø73's are used on the same bus, whenever one executes an INC (X) or DEC (X) instruction, other processors must remain idle. These instructions are used, generally, for communications between processors in a multiprocessor system.

2.3 Statements

2.3.1 INPUT Statement

Data can be input to an NSC Tiny BASIC program by using the INPUT statement. One or more items (variables, expressions etc.), separated by commas, may be entered according to the following formats:

```
        1Ø INPUT A
        2Ø INPUT B,C
```

When the statement at Line 1Ø is executed, NSC Tiny BASIC prompts the user with a question mark. The user types in a number which is assigned to the variable A after the RETURN key is pressed. NSC Tiny BASIC then prompts the user with another question mark. The user types in two expressions, separated by commas, which will be assigned to B and C in that order.

```
        RUN
        ? 45
        ? 237, 4455
```

NSC Tiny BASIC would now continue with execution of the program. String input is also allowed. See the String Handling section in this chapter for more information.

NSC Tiny BASIC accepts both numbers and expressions typed in response to an INPUT request. For example:

```
        >10 A=10
        >20 INPUT B,C
        >30 PRINT B,C
        >RUN
        ?A+1,A*2
         11 20
```

The comma between the entered expressions is not mandatory and can be replaced by spaces if the second expression does not start with a plus or minus sign.

There must be at least as many expressions in the input list as variables in the INPUT statement. If an error occurs when NSC Tiny BASIC tries to evaluate the typed-in expression, the message:

```
        RETYPE
```

is printed along with the error message, and the question mark (?) prompt will appear again so that the user can type the expressions correctly.

The correct response to an 'INPUT $factor' statement is a string, terminated by a carriage return. Quotation marks are not used for input.

INPUT may not be used in the command mode.

2.3.2 PRINT Statement (Output)

The PRINT Statement is used to output information from the program. Quoted strings are displayed exactly as they appear with the quotes removed. Numbers are printed in decimal format. Positive numbers will be preceded by a space, and negative numbers will be preceded by a minus (-) sign. There is a trailing space for all numbers. A semicolon (;) at the end of a PRINT Statement suppresses the usual carriage return and line feed with which NSC Tiny BASIC terminates the output.

Strings stored in memory (such as those generated by a String Input Statement) may also be printed. Refer to the String Handling Section in this chapter for more information. Typical example:

```
        >PRINT "THIS IS A STRING"
        >20 A=10
        >30 B=20
        >40 PRINT "10 PLUS 20=", A+B
        >RUN
        THIS IS A STRING
        10 PLUS 20=30
```

2.3.3. LET Statement (Assignment)

The word, LET, may be used or omitted in an Assignment Statement. The execution of an assignment statement is faster if the word LET is used. The left portion of an Assignment Statement may be a simple variable (A-Z), STAT or a memory location indicated by an @ followed by a variable, number or an expression in parentheses, (refer to Indirect Operator for more information). Examples:

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

Conditional assignments may be made without using an IF statement. The method hinges on the fact that all predicates are actually evaluated to yield -1 if true, and 0 if false. Thus, if a predicate is enclosed in parentheses, it may be used as a multiplier in a statement as:

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

which would assign the absolute value of A to X.

2.3.4 The GO TO Statement

NSC Tiny BASIC allows GO TO Statements to allow program branches to a specific line number or a line number called by an arbitrary expression. As examples:

```
        10 GO TO 50
```

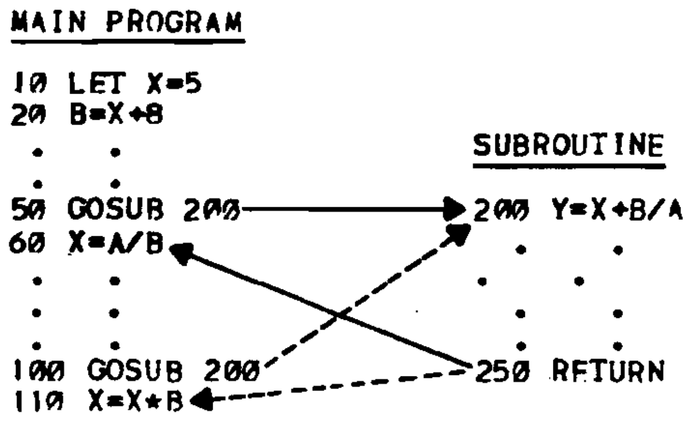
would cause the program to jump from Line 10 directly to Line 50, but

```
        10 GO TO X+5
```

would cause the program to jump from Line 10 to Line X+5. Thus, the value of X is variable allowing dynamic control of program execution at this point.

2.3.5 GOSUB/RETURN Statements

These instructions are useful when a computation or operation must be performed at more than one place in a program. Rather than write the routine at each place where needed, a GOSUB instruction is used to "call" the computation or operation (referred to as a SUBROUTINE). After the subroutine has been executed, a RETURN instruction (the last instruction of the subroutine) causes the program to resume execution at the next line number following the original GOSUB instruction. As an example:



```
MAIN PROGRAM

10 LET X=5
20 B=X+8
 .   .                            SUBROUTINE
 .   .
50 GOSUB 200 ---------------->200 Y=X+B/A
60 X=A/B                          .   .
 .   .                            .   .
 .   .                            .   .
 .   .                            .   .
100 GOSUB 200                   250 RETURN
110 X=X*B
```

On the first GOSUB call, the order of execution follows the solid arrows. At the second GOSUB call (Line 100), the order of execution follows the dashed arrows.

NOTE: GOSUBs may be nested up to 8 levels deep (including interrupt levels).

2.3.6 IF/THEN Statement

This instruction allows for program control to be modified by a logical test condition. The test condition follows the IF clause of the statement. When the test condition is true (non-zero), the THEN portion of the statement will be executed. When the test condition is false (zero), the THEN portion is ignored and execution continues at the next numbered line of the program.

```
50 IF X>J THEN GO TO 140
```

NSC Tiny BASIC allows the omission of the word THEN from an IF/THEN Statement. This omission, also allowed on some larger BASICs, enhances the clarity of the program. The above statements then become:

```
50 IF X>J GO TO 140
```

2.3.7 DO/UNTIL Statements

This instruction is not available in standard BASICs. This statement is used to program loops, keeping GO TO statements to a minimum. The overall effect is to greatly improve readability and clarity of NSC Tiny BASIC programs. The following example shows the use of DO/UNTIL Statements to print numbers less than 100:

```
10 PRINT 1: PRINT
20 PRINT 2
30 I=3                    :REM I IS NUMBER TESTED
40 DO
50 J=I/2                  :REM J IS THE LIMIT
60 N=1                    :REM N IS THE FACTOR
70 DO                     :REM SEEKS A DIVISIBLE FACTOR OF I
80 N=N+2
90 UNTIL (MOD(I,N)=0 OR (N>J))
100 IF N>J PRINT I        :REM NO DIVISIBLE FACTOR
110 I=I+2
120 UNTIL (I>100)         :REM ENDS THE SEARCH
```

By enclosing a zero or more statements between the DO and the UNTIL <condition> statement (where the <condition> is any arbitrary expression), the statements between will be repeated as a group until the <condition> evaluates to a non-zero number (a true condition). DO/UNTIL loops can be nested, and NSC Tiny BASIC will report an error if the nesting level becomes too deep, (more than eight levels).

2.3.8 FOR/NEXT Statements

These statements are identical to the FOR/NEXT Statements in standard BASICs. The STEP in the FOR statement is optional. If it is not included, a STEP value of +1 is assumed. The value of the STEP may be either positive or negative. Starting and ending values of the FOR/NEXT loop are included in the FOR statement. The loop is repeated when the NEXT statement has been executed provided the upper limit of the FOR statement has not been reached. When the upper limit is reached, the program will exist from the FOR/NEXT loop. NSC Tiny BASIC causes an error break if the variable in the NEXT statement is not the same variable as that used in the FOR statement.

FOR/NEXT loops may be nested, and NSC Tiny BASIC will report an error if the nesting level becomes too deep: a depth of four levels of FOR loop nesting is allowed. A FOR loop will be executed at least once, even if the initial value of the control variable already exceeds its bounds before starting. The following program would do nothing but print the odd integers less than 100.

```
10 N=100                  :REM UPPER LIMIT
20 FOR I=1 TO N STEP 2    :REM START AT 1 WITH STEP OF 2
30 PRINT I                :REM PRINT A NUMBER
40 NEXT I                 :REM REPEAT (at Line 20)
```

2.3.9 LINK Statement

Control may be transferred from an NSC Tiny BASIC program to an INS 8073 machine language routine by means of a LINK Statement. This allows the user to make use of routines which may be more efficiently performed in machine language. A statement of the form LINK <address> will cause control to be transferred to the INS8073 machine language routine starting at <address>. Control is transferred by execution of a JSR instruction. The pointers may be modified by the routine. P3's value is unpredictable, and P2 points at the start of A-Z variable storage. Variables are stored in alphabetically ascending order, two bytes each, low order byte first then high-order byte.

Example:

```
>10 LINK #1800                          NSC Tiny BASIC transfers to
>20 IF A=0 THEN PR "SENSE A IS LOW"       address #1800 to read
>30 IF A=1 THEN PR "SENSE A IS HIGH"      sensor.
>99 STOP                                Program transfers back to
>RUN                                      NSC Tiny BASIC
SENSE A IS HIGH

STOP AT 99
>RUN
SENSE A IS LOW

STOP AT 99

1               .TITLE SENSE
2 0000          .=01800          ;HEXADECIMAL
3 1800 06       LD    A,S
4 1801 D410     AND   A,=16
5 1803 6C02     BZ    LOW
6 1805 C401     LD    A,=1
7 1807 CA00     ST    A,0,P2     ;STORES ACCUMULATOR INTO LOCATION
8 1809 5C       RET                    OF VARIABLE A
9         0000  .END
```

2.3.10 ON Statement

This statement is used for processing interrupts. The format of the statement is:

    ON interrupt-#1, line-number

When numbered interrupt (interrupt-#) occurs, NSC Tiny BASIC executes a GOSUB statement beginning at line number "line-number". If "line-number" is zero, the corresponding interrupt is disabled at the software level. Interrupt numbers may be 1 or 2. Use of the ON statement disables console interrupts (BREAK function). Interrupts must also be enabled at the hardware level by setting the Interrupt Enable bit in the status register (for example, using STAT=1).

2.3.11 STOP Statement

Although the last line of a program does not need to be a STOP statement, it is a useful debugging tool for programs. The STOP statement may be inserted as breakpoints in an NSC Tiny BASIC program.

When NSC Tiny BASIC encounters a STOP statement, it prints a stop message and the current line number. It then returns to the edit mode. Thus, the programmer can see whether his program reached the desired point. Any number of STOP statements may appear in the program. By removing the STOP statements, one by one, a program can be tested by parts until the debugging process is completed.

Execution of a stopped program may be continued after the STOP by a CONT (continue) command.

2.3.12 DELAY Statement

This statement delays NSC Tiny BASIC for "expr" time units (nominally milliseconds, 1-1040). Delay 0 gives the maximum delay of 1040 milliseconds. The format is:

```
        DELAY expr
```

Example:

```
        >10 DELAY 100          Delay 100 milliseconds.
```

2.3.13 CLEAR Statement

This statement initializes all variables to 0, disables interrups, enables BREAK capability from the console, and resets all stacks (GOSUB, FOR-NEXT, DO-UNTIL).

Example:

```
        >10 ON  2,250         Break is disabled, Interrupt 2 is enabled.
          .
          .
          .
          .
          .
        >300 CLEAR            Break is re-enabled, Interrupt 2 is disabled.
```

2.4 Indirect Operator

The Indirect Operator is an NSC Tiny BASIC exclusive, at least in the realm of BASIC. It accomplishes the functions of PEEK and POKE with a less cumbersome syntax. The Indirect Operator is a way to access absolute memory location although its applications are not limited to that. Its utility is especially significant for microprocessors, such as the INS8073, where interfacing is commonly performed through memory addressing.

An "at" sign (@) which preceeds a constant, a variable or an expression in parentheses 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 an input device has an address of #6800 (hexadecimal), the statement:

```
        LET X=@#6800
```

would input from that device and assign the value of the input to the variable X. If the address of an output device was #6801, the statement:

```
        @#6801=Y
```

would output the least significant byte of Y to the device.

The indirect operator accessing memory locations only one byte at a time. An assignment such as @A=248 changes the memory location pointed to by A to 248 (1111 1000) binary, since 248 can be expressed as one byte. However, an assignment such as @A=258 changes the memory location pointed to by A to 2 because the value of 258 cannot be expressed by a single byte, as shown below:

```
        258    = 1 0000 0010
           10    ▲ └────┬───┘
                 │ ┌────┘
extra bit────────┘ └one byte (stored into location to which A would point)
```

Only the least significant byte of 258 (which is 2) is stored at that location. The extra bit would be lost forever.

Any place that a variable, such as B, would be legal, the construct "@B" would also be legal. The meaning of @B is: the byte located at the memory location whose address is the value of B. Other examples:

```
40 LET B=6000         Assigns 6000 to B.
50 LET @ B=100        Stores decimal 100 in memory location 6000.
60 LET C=@B           Sets C=to 100.
70 PRINT @6000        Prints 100.
80 LET D=@(A+10*D)    Sets D=the value stored in memory location
                      (A+10*D).
```

Parentheses are required when applying @ to an expression.

2.5 Multiple Statements On A Line

More than one statement can be placed on one program line. This is accomplished by placing a colon between the statements. Readability of the program can be improved, and memory space can be saved by using this technique. As an example of the use of multiple statements:

```
200 PRINT "MY GUESS IS",Y:PRINT "INPUT A POSITIVE NUMBER"::
                 INPUT X:IF X <=0 GO TO 200
```

If X is negative or zero, the user will be instructed to enter a positive number, and the program returns to Line 200 for a new guess. If the user had entered a positive number correctly, the program would have proceeded to the next numbered line after Line 200.

Care in use of multiple statements per line must be exercised. The above example shows that if the condition of the IF STATEMENT is false, control is passed to the next program line. Anything else on the line containing multiple statements will be ignored.

2.6 String Handling

String inout may be accomolished by executing a statement of the form INPUT $ F, where F is a Factor syntactically (see Grammar). When the orogram reaches this statement during program execution, NSC Tiny BASIC promots the user with a question mark (?). All line editing characters may be used (back space, line delete, etc.). If a control-U is tyoed to delete an entered line, NSC Tiny BASIC will continue to prompt for a line until a line is terminated by a carriage return. The line is stored in consecutive locations starting at the address oointed to by F, up to and including the carriage return. Example:

```
        20 INPUT $ A    may also be written    20 INPUT $A

        and inputs a string to successive memory locations starting
            at A.
```

2.6.1 String Outout

An item in a PRINT statement can include a string variable in the form of $B, where B is a factor. When the print statement is encountered during program execution, the string will be printed beginning at the address B up to, but not including, a carriage return. A keyboard interrupt will also terminate the printing of the string if detected before the carriage return. Example:

```
        50 PRINT $B    prints the string beginning at the location
                                 pointed to by "B".
```

2.6.2 String Assignment

String variables can be assigned to characters in quotes just as other variables are assigned numerical values. A statement of the form $C= "THIS STRING IS A STRING" (when encountered during program execution) would cause the characters in quotes to be stored in memory starting at the address indicated by C up to and including the carriage return. Example:

```
        70 $D="THIS IS A STRING WITH NO INPUT STATEMENT."
        A "T" is stored at location "D", and H at location "D+1" etc.
```

2.6.3 String Move

Strings can be moved from one memory block to another memory block using this feature. A statement of the form $A=$B (where A and B are Factors) will transfer the characters in memory beginning with the address B to the memory beginning with address A. The last character, normally a carriage return, is also copied. Also, a statement such as $(A+1)=$A would be disasterous since it causes the entire contents of the RAM to be filled with the first character of $A.

2.6.4 String Examples

```
10 A=TOP                 :REM A POINTS TO EMPTY RAM ABOVE TOP OF
                          PROGRAM
20 C=TOP+100             :REM C POINTS TO RAM 100 BYTES ABOVE A
30 D=TOP+200             :REM D POINTS TO RAM 100 BYTES ABOVE C
40 INPUT $A              :REM STORES CHARACTERS WHERE A POINTS
50 PRINT $A
60 LET $C= "IS THE STRING INPUT AT LINE 10"
70 $D=$C                 :REM STORES CHARACTERS WHERE D POINTS
80 PRINT $D
```

2.7 Commands

2.7.1 NEW expr

This command establishes a new start-of-program address equal to the value of "expr". NSC Tiny BASIC then executes its initialization sequence which clears all variables, resets all hardware/software stacks, disables interrupts, enables BREAK capability from the console, and performs the nondestructive RAM search described in part one. If the value of "expr" points to a ROM address, the NSC Tiny BASIC program which begins at this address will be automatically executed. Program memory (including the end-of-program pointer used by the editor) is not altered by this command.

Example:

```
>NEW 1000
```

NEW used without an argument sets the end-of-program pointer equal to the start-of-program pointer so that a new program may be entered. If a program already exists at the start-of-program address, it will be lost.

Example:

```
>NEW 1000       Sets program pointer to 1000
 NEW            Sets end-of-program pointer to 1000
```

2.7.2 RUN

Runs the current program.

Example:

```
>RUN    Execution begins at lowest line number
```

2.7.3 CONT

Continues execution of the current program from the point where execution was suspended (via a STOP, console interrupt, or reset). An NSC Tiny BASIC program that is executing can be interrupted by pressing the BREAK or RESET keys on the keyboard. Execution can be resumed by entering the CONT command.

Example:

```
>RUN
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10  Press  BREAK or RESET.
^C
>CONT
 THIS IS THE STRING INPUT AT LINE 10
 THIS IS THE STRING INPUT AT LINE 10
 And so on...
```

2.7.4 LIST (expr)

Lists the current program (optionally starting at the line number specified by (expr)).

Example:

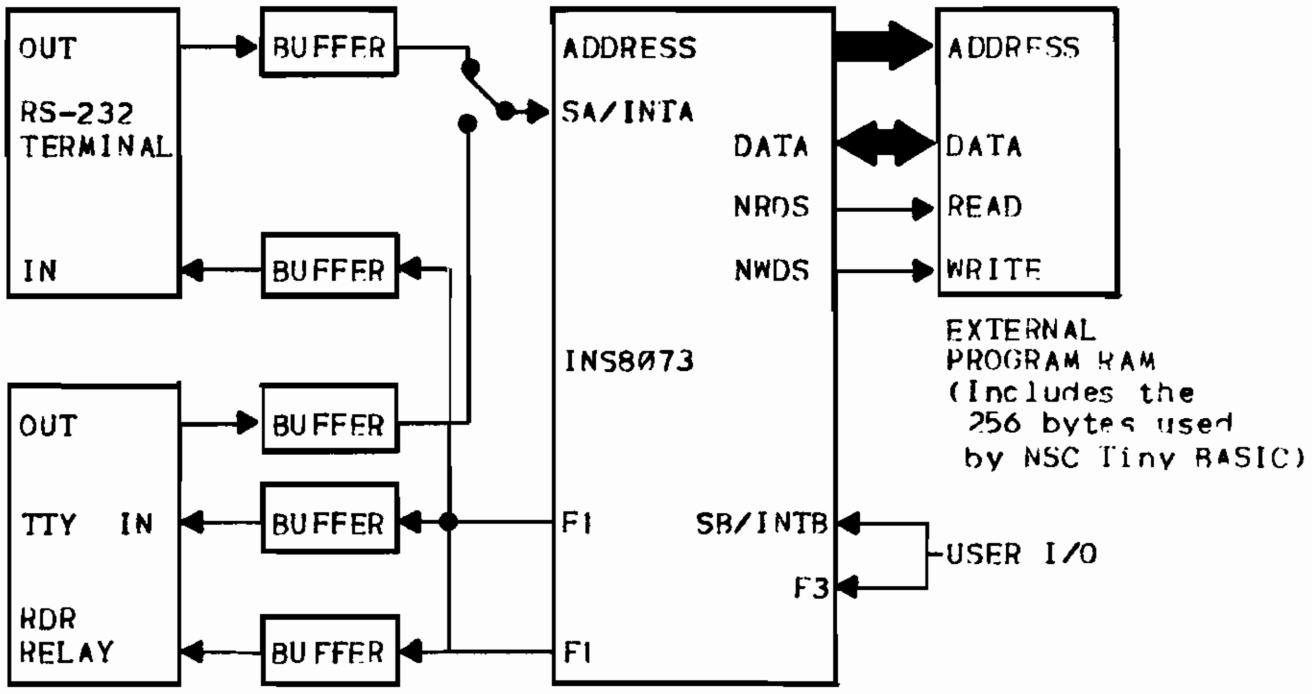
```
>LIST 10

10 INPUT $A
20 PRINT $A
30 LET $C="IS THE STRING INPUT AT LINE 10"
40 $D=$C
50 PRINT $D
```

Section 3

CHAPTER 1

1.1 Introduction

The design of an INS8073-based system is quite straightforward. Figures 1-1 through 1-3 illustrate this point. Figure 1-1 shows a minimum size RAM-based system: this is the kind of system used in engineering labs for software development. For stand-alone program operation a system like the one shown in Figure 1-2 can be used, provided 256 bytes of RAM are available for variable storage. Figure 1-3 is an expansion of this system to allow a 32-bit parallel I/O interface.



NOTE: It is not necessary to have a TTY and an RS-232 terminal. Either one may be omitted.

Figure 1-1. Minimum RAM-Based System

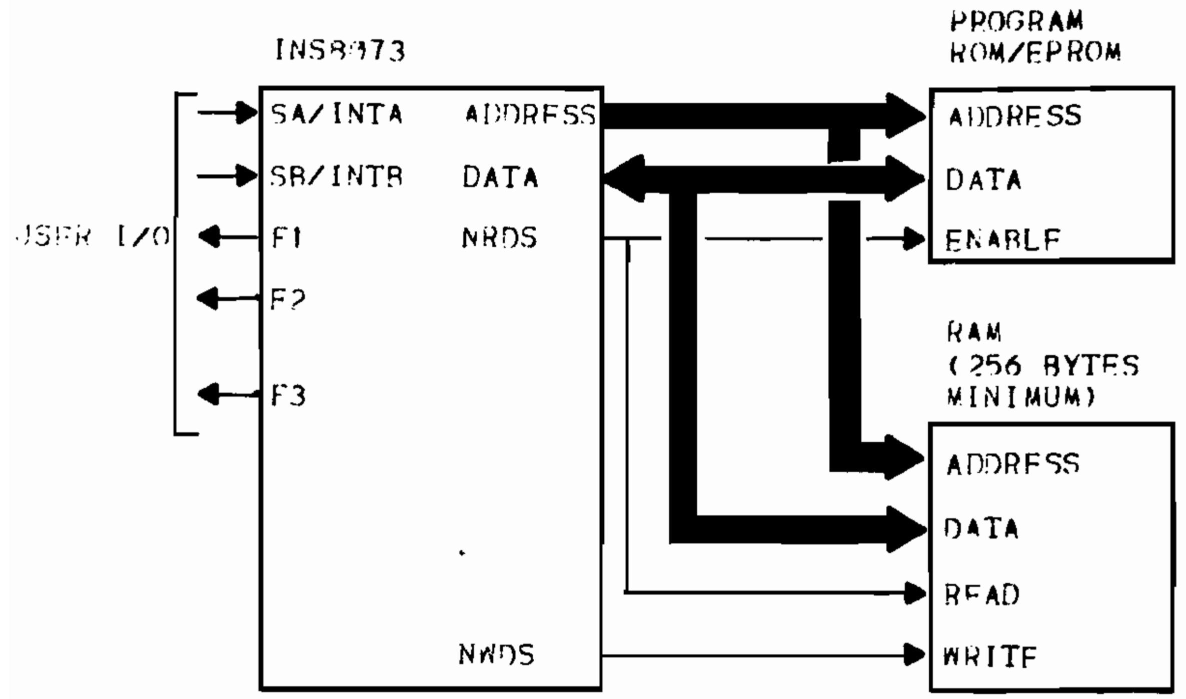


Figure 1-2. Minimum ROM/EPROM-Based System

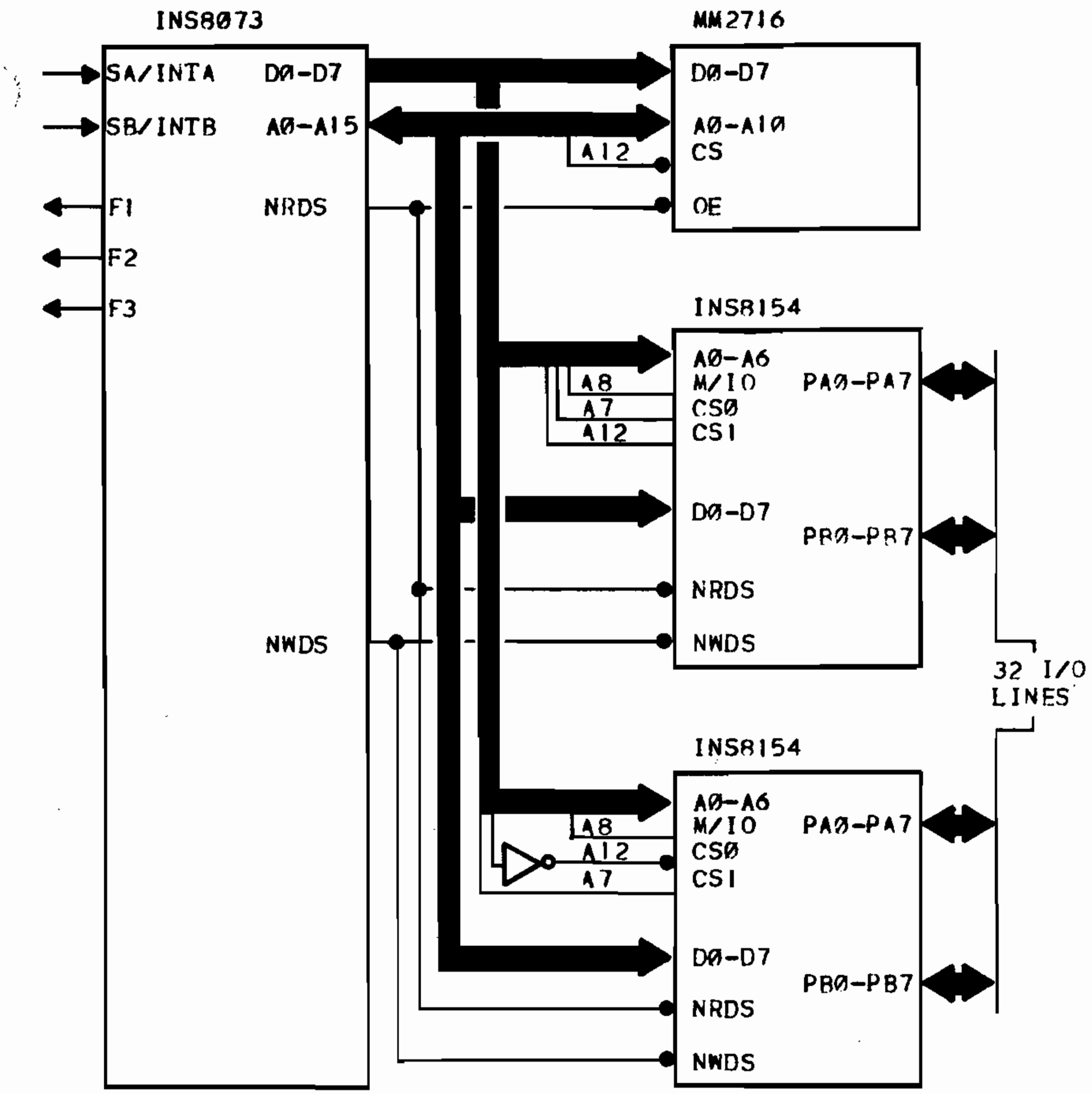


Figure 1-3. I/O Expansion of the Minimum ROM-Based System

1.2 An NSC Tiny BASIC Example System, Functional Specification

It is obvious, from the preceding examples, that by using only a small number of ICs, an extremely powerful and flexible system can be easily developed. To illustrate this point, we will design a system to satisfy all of the following requirements:

1. To allow the user to enter, debug and execute RAM-based NSC Tiny BASIC programs up to 130 lines in length.

2. To interface to a terminal or TTY for program entry and debug. Multiple data rates (110, 300, 1200 and 4800 Baud) should be supported.

3. To allow the user to transfer RAM resident programs into EPROM.

4. To allow an EPROM program to be run in a real-time control applications where a terminal is not present.

5. To have ample I/O capability flexible enough to interface to most user systems.

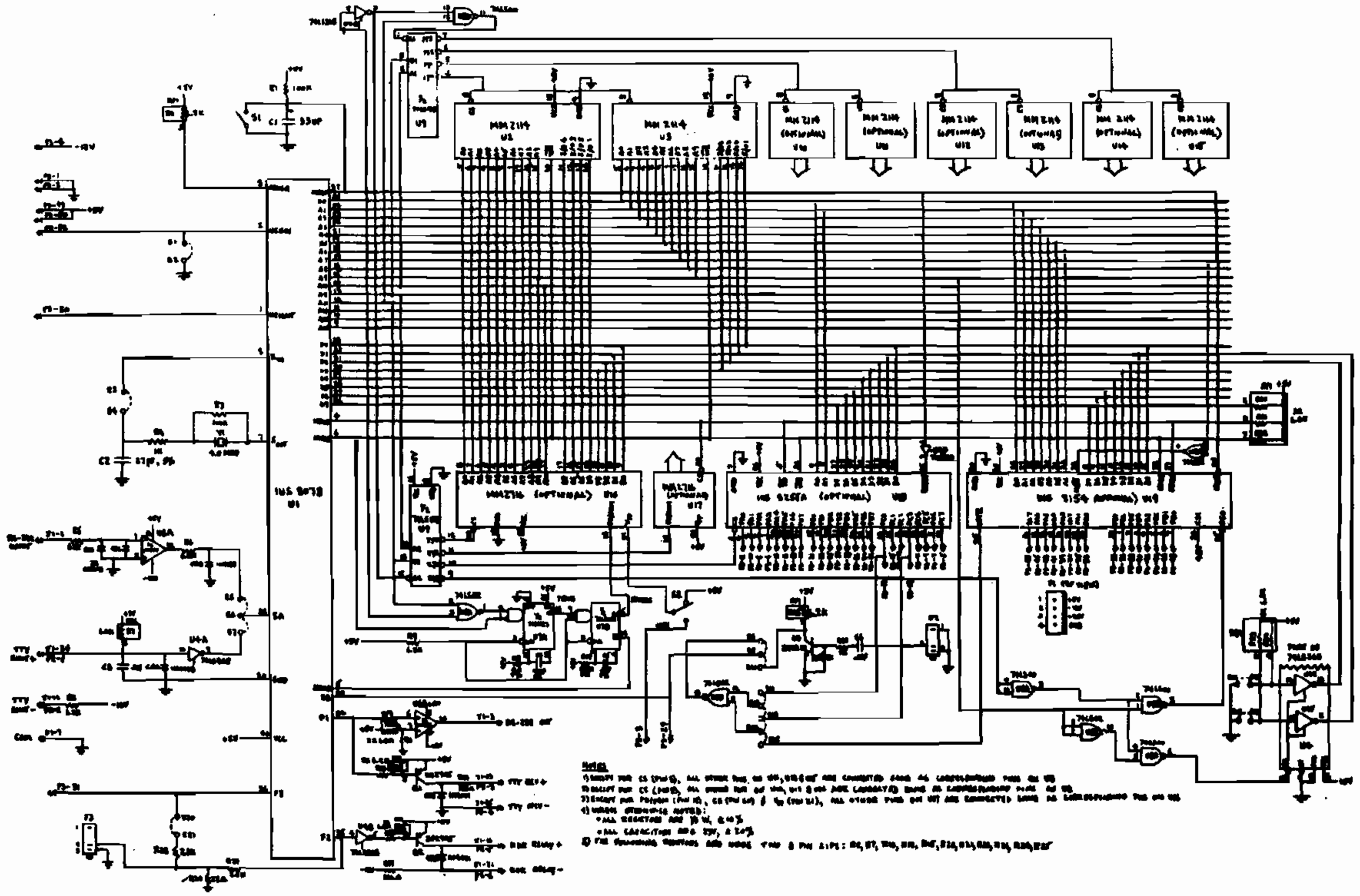
6. To provide the user with "scratchpad" RAM for use when assembly language subroutines are invoked via the "LINK" statement

7. To support at least two interrupts.

8. To fit the entire system on a single 5" x 7" PC card.

9. To satisfy all design requirements using a minimum number of IC's. Expansion of the minimum system should be accomplished by simple addition of "optional" RAM, EPROM and I/O devices on the PC card.

Although meeting all of the above requirements may at first seem difficult, these objectives are easily attainable, as the following paragraphs will show.



NOTES:
1) EXCEPT FOR CS (PIN 8), ALL OTHER PINS ON U5, U12 & U13 ARE CONNECTED SAME AS CORRESPONDING PINS ON U2
2) EXCEPT FOR CS (PIN 8), ALL OTHER PINS OF U10, U11 & U16 ARE CONNECTED SAME AS CORRESPONDING PINS OF U3
3) EXCEPT FOR PROGRAM (PIN 18), CS (PIN 20) & V_{PP} (PIN 21), ALL OTHER PINS ON U17 ARE CONNECTED SAME AS CORRESPONDING PINS ON U6
4) UNLESS OTHERWISE NOTED:
 • ALL RESISTORS ARE 1/4 W, ±10%
 • ALL CAPACITORS ARE 25V, ±20%
5) THE FOLLOWING [illegible] ARE MADE [illegible]: [illegible]

1.3 Hardware Design of a Small INS8073-Based System

A system that meets all of the above design requirements is shown in Figure 1-4. The type, designation and function of each IC shown is as follows:

IC TYPE	IC DESIGNATOR	FUNCTION
INS8073	U1	NSC Tiny BASIC processor.
MM2114	U2, U3	U2 and U3 provide 1K bytes of static RAM. (Each MM2114 provides 1Kx4 bits.)
74LS368	U4A	Inverter for TTY input interface.
	U4B	Inverter for TTY reader relay interface.
	U4C	Inverter for RAM address mapping logic.
	U4D	Inverter for power-on reset of INS8255A.
	U4E, U4F	TRI-STATE inverters for selection of multiple Baud rates.
74LS02	U5A	Two input NOR gate. Used for address mapping of the EPROM programmer.
	U5B	Two input NOR gate. Used to select interrupt source(s) to INS8073.
	U5C	Two input NOR gate. Used in Baud rate selection logic.
	U5D	Two input NOR gate. Used for address mapping of the INS8154.

LM747	U6A	The LM747 is a dual OP amp. U6A buffers the positive/negative voltage levels received from the RS-232 compatible input to the TTL levels required by the INS8073.
	U6B	U6B buffers the TTL levels generated by the INS8073 to the positive/negative voltage levels required to drive the RS-232 compatible output.
74LS123	U7A	The 74LS123 is a dual One-shot. U7A provides adequate address/data setup time to program the MM2716 EPROM.
	U7B	U7B provides the 50 msec programming pulse required to write data into the MM2716 EPROM.
74LS00	U8A,B,C	U8 is a quad NAND gate. U8A, U8B and U8C are used in the Baud rate selection logic.
	U8D	Used in the RAM address mapping logic.
74LS139	U9	Dual 2 line to 4 line decoder with active low outputs. Provides address mapping for RAM, EPROM and I/O ICs.
MM2114	U10-U15	Provide an additional 3K bytes of optional RAM program memory.
MM2716	U16,U17	Provide up to 4K bytes of optional EPROM program memory. (Each MM2716 contains 2K bytes.)
INS8255A	U18	Optional Programmable Peripheral Interface chip. Provides 24 I/O lines that may be used to interface with the user's system. I/O pins may be programmed as inputs, outputs or bidirectional, including the required handshake signals. (Refer to the INS8255A Data Sheet for additional information.)

INS8154	U19	Optional 128 byte RAM-I/O chip. Provides 128 bytes of scratch-pad RAM for use in assembly language subroutines. Also provides 16 I/O lines that may be individually programmed as input or output, including strobe mode with handshake. (Refer to INS8154 Data Sheet for additional information.)

Note from the above tabulation that the minimum system consists of only nine IC's U1 - U9. Together they provide 1K bytes of RAM program memory, an RS-232/TTY interface, an MM2716 EPROM programmer, automatic Baud rate selection and complete decoding for the fully expanded system. The fully expanded system consists of 19 IC's.

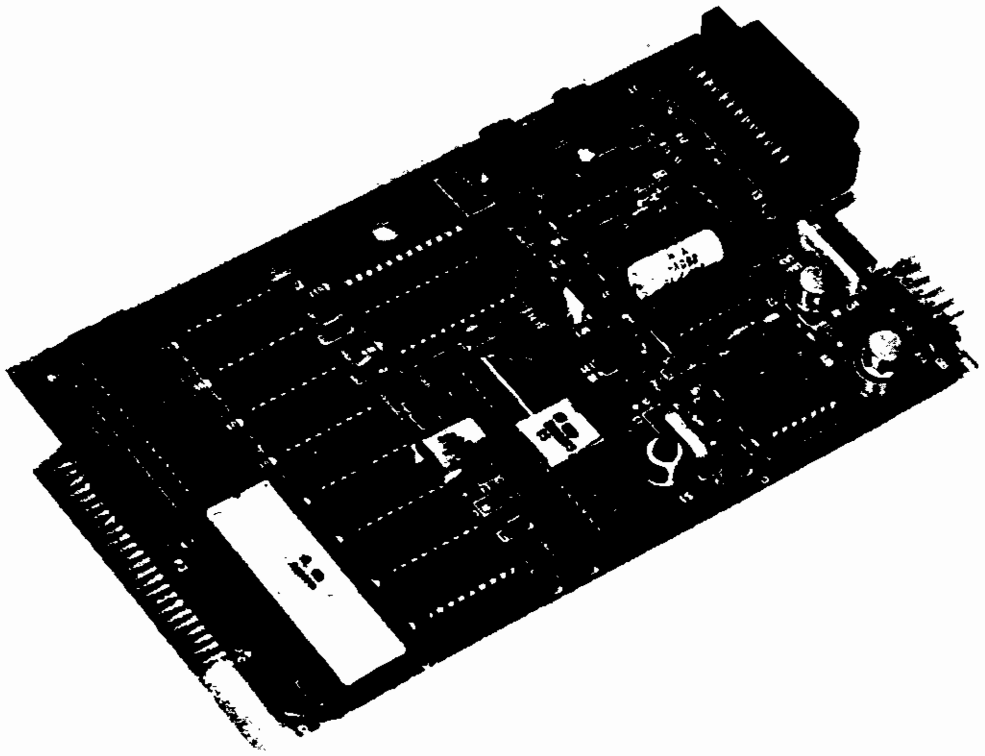


Figure 1-5 Photo of NSC Tiny BASIC Card

1.4 Addressing Requirements/Capabilities of Each System Component

Each of the system components shown in Figure 1-4 must be assigned to address locations in memory. The built-in address decoding capability of each system component can be summarized as follows:

4K Bytes of RAM

Each of the four pairs of MM2114 chips fully decodes 10 bits and can be selected via one active low select per pair.

4K Bytes of EPROM

Each of the two MM2716 EPROMSs fully decodes 11 bits and provides two active low select lines per device for reading of data.

INS8255

The INS8255 contains three I/O ports and one control word register, all of which are decoded on chip via two address input lines. The device is enabled via a single active low select line.

INS8154

The INS8154 contains 128 bytes of RAM, two I/O ports and two data direction registers, all of which are decoded on chip via eight address lines. The device is enabled via one active high select line and one active low select line.

Baud Rate Selection Logic

The INS8073 selects the Baud rate by reading the contents of memory location X'FD00. To program the Baud rate, this location must be decoded via external logic, and the appropriate logic levels supplied on data lines 1, 2 and 7. (Refer to RS-232/Current Loop Interface section for additional details.)

EPROM Programmer

To program an MM2716 EPROM, address/data are supplied by the INS8073 to the 2716 socket U16 in Figure 1.4. When VPP = +25V and address/data are valid, a single byte may be written by providing a 50 msec programming pulse to pin 18 while the chip is deselected via a logic 1 on pin 20. A byte which has been written may be subsequently read by simply supplying the correct address and providing a logic 0 on pin 20. (Refer to MM2716 Data Sheet for additional details.)

1.5 Memory Mapping Constraints For All System Components

The components described above can be mapped into memory in a variety of ways. The system constraints imposed upon this mapping are the following:

1. The decoding hardware will be implemented using a minimum number of ICs. This implies that the system components will be only partially decoded, resulting in multiple images of each component in memory.

2. Although multiple memory images of each system component may be present, the mapping hardware will be designed such that it is impossible to enable more than one system component at a time. This restriction eliminates the possibility of causing data bus conflict as the result of a programming error. (A data bus conflict could cause transmission/receipt of invalid data and chip damage.)

3. NSC Tiny BASIC program RAM will be decoded as a contiguous block so that the INS8073 can successfully identify the beginning and the end of the program RAM that is actually present.

4. The RAM and the I/O ports of the INS8154 will be located in the address range X'FF00 - X'FFBF. This allows INS8073 assembly language subroutines to address the INS8154 using the DIRECT addressing mode. (Use of DIRECT addressing eliminates the need to dedicate or multiplex a pointer in order to address the INS8154. For additional details on DIRECT addressing, refer to the INS8070 Data Sheet.)

5. When on-card EPROM is present, it will be located starting at address X'8000. This allows the system to be used in real-time control applications where a terminal is not present.

All of the above constraints are satisfied by the memory assignment shown in Figure 1-5 and Figure 1-6. Figure 1-5 shows how the 64K addressing space of the INS8073 is to be partitioned. Figure 1-6 shows the address bits (in boldface) that are actually decoded by the hardware shown in Figure 1-5, resulting in multiple (but not overlapping) memory images of each component. The locations of these multiple images are also shown, with address bits A12 - A15 specifying one of 16 possible memory "pages", each of which contains 4K bytes.

1.6 System Generated Interrupts

NSC Tiny BASIC supports interrupts via the "ON" statement. As shown in Figure 1-5, interrupts generated by the INS8154 and/or INS8255 may be connected, at the user's discretion, to the SB/INTB pin of the INS8073. When this is done the INS8073 SB/INTB pin may be used to detect interrupts under control of the user's program. If interrupts are disabled, the SB/INTB pin may be employed as a sense pin that can be examined via the NSC Tiny BASIC "STAT" Function or the "ON" Statement.

HEX ADDRESS	MEMORY CONTENTS
ØØØØ-Ø9FF	INS8Ø73 ON-CHIP NSC TINY BASIC INTERPRETER
⋮	⋮
1ØØØ-13FF	RAM Ø (1K BYTES)
14ØØ-17FF	RAM 1 (1K BYTES)
18ØØ-1BFF	RAM 2 (1K BYTES)
1CØØ-1FFF	RAM 3 (1K BYTES)
2ØØØ-27FF	MM2716 EPROM PROGRAMMER
⋮	⋮

HEX ADDRESS	MEMORY CONTENTS
8ØØØ-87FF	ROM Ø (2K BYTES)
88ØØ-8FFF ⋮	ROM 1 (2K BYTES) ⋮
F7ØØ-F7Ø3	INS8255A
⋮	⋮
FDØØ	BAUD RATE SELECT
⋮	⋮
FFØØ-FF7F	INS8154 RAM (128 BYTES)
FF8Ø-FFA4	INS8154 I/O PORTS/CONTROL
⋮	⋮
FFCØ-FFFF	INS8Ø73 ON-CHIP RAM (64 BYTES)

Figure 1-6 Partitioning of the INS8Ø73 64K Addressing Space

ADDRESS BITS																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	Ø	
Ø	X	X	Ø	Ø	X	X	X	X	X	X	X	X	X	X	X	EPROM PROGRAMMER (X'2ØØØ-X'27FF)
Ø	X	X	1	Ø	Ø	X	X	X	X	X	X	X	X	X	X	RAM Ø (X'1ØØØ-X'13FF)
Ø	X	X	1	Ø	1	X	X	X	X	X	X	X	X	X	X	RAM 1 (X'14ØØ-X'17FF)
Ø	X	X	1	1	Ø	X	X	X	X	X	X	X	X	X	X	RAM 2 (X'18ØØ-X'1XFF)
Ø	X	X	1	1	1	X	X	X	X	X	X	X	X	X	X	RAM 3 (X'1CØØ-X'1FFF)
1	X	X	Ø	Ø	X	X	X	X	X	X	X	X	X	X	X	ROM Ø (X'8ØØØ-X'87FF)
1	X	X	Ø	1	X	X	X	X	X	X	X	X	X	X	X	ROM 1 (X'88ØØ-X'8FFF)
1	X	X	1	Ø	X	X	X	X	X	X	X	X	X	X	X	INS8255A (X'F7ØØ-X'F7Ø3)
1	X	X	1	1	X	Ø	X	X	X	X	X	X	X	X	X	BAUD RATE SELECT (X'FDØØ)
1	X	X	1	1	X	1	X	X	X	X	X	X	X	X	X	INS8154 RAM (X'FFØØ-X'FF7F)
1	X	X	1	1	X	1	X	1	X	X	X	X	X	X	X	INS8154 I/O PORTS (X'FF8Ø-X'FFA4)

NOTES: 1. "X" refers to an address bit that may be zero or one.
2. Bits that are actually decoded by the hardware shown in Figure 1-4 appear in boldface type.

Table 1-1. Address Bits

Decoding only the indicated address bits results in the following multiple memory images of each component. This list is organized in three columns. The first column shows the component, the second shows the page in memory into which that component is mapped, (page numbers range from 0 to F, each page being 4K bytes): and the third shows how the elements of a shared page are subdivided.

COMPONENT	PAGES	ADDITIONAL CONDITIONS
EMPROM PROGRAMMER	0, 2, 4, 6	
4K RAM	1, 3, 5, 7	
4K ROM	8, A, C, E	
INS8255A	9, B, D, F	A11 = 0
BAUD RATE SELECT	9, B, D, F	A11 = 1, A9 = 0
INS8154	9, B, D, F	A11 = 1, A9 = 1

Figure 1-6 Address Bit Decoding for the System

1.7 RS-232/Current Loop Interface

The described Baud rate is automatically selected when the INS8073 is initialized, or when a "NEW" command is issued. Initialization is automatically accomplished at VCC power-on by R1 and C1 in Figure 1-5. (Pressing switch S1 also causes the INS8073 to be initialized.) The Baud rate is jumper selectable as follows:

BAUD RATE	E16-E17 JUMPER	E18-E19 JUMPER	D7	D2	D1
110	PRESENT	PRESENT	1	1	1
300	PRESENT	ABSENT	1	1	0
1200	ABSENT	PRESENT	1	0	1
4800	ABSENT	ABSENT	1	0	0

If only the 110 Baud rate is required, pullup resistors on data lines D1, D2 and D7 represent the only external hardware required to select this rate.

As shown in Figure 1.5, the INS8073 F1 flag is double buffered to provide an RS-232 compatible voltage output and a 20ma current output. Positive and negative RS-232 levels are generated by the LM 747 op amp. The 20 ma current drive is produced by transistor switch Q1 and Resistor R15.

The INS8073 R2 flag is used to enable/disable the TTY reader relay via transitor switch Q2 and current limiting register R2. These components will supply 20 ma of current to a 12V (600) relay.

The INS8073 will accept serial ASCII input data in its SA/INTA input. As shown in Figure 1.5, the RS-232 input signal is selected via a jumper between E5-E6, or the TTY input signal may be selected via a jumper between E6-E7.

CHAPTER 2

2.1 MM2716 EPROM Programming Software

An NSC Tiny BASIC utility program that programs MM2716 EPROMs, and one that will work with the system shown in Figure 1-4 is shown in Appendix D. A PROM with this program must be plugged into socket U17 to operate these utility programs. The programming software is called from NSC Tiny BASIC by typing:

```
>NEW #8800
```

This program decodes and executes the following ten commands:

```
COPY
PROGRAM
VERIFY
ERASE CHECK
FILL
DUMP
LOAD
ASCII LOAD
WRITE
READ
```

Each command is designated by a single command letter followed by 1-3 address and/or data fields. The user is prompted for a command input by the message "COM?". In response to this, a legal command in the proper format must be entered. If an illegal command letter or improper format are employed, the user will be prompted to re-enter another command by the message: "INPUT ERROR; TRY AGAIN." Addresses and data should be entered in hexadecimal, without the preceding "#" sign. Address and data fields should be delimited by slashes (/) or by commas (,). Spaces are optional and are ignored. For convenience, "default" addresses and/or data are associated with each command. These default values allow the user to enter only the command letter, followed by a carriage return. When this is done the default values are substituted for the address/data that was not entered. The default values are preset to the most commonly used address/data for each command. When the default values are unsuitable, the desired address/data must be entered.

The commands are discussed in detail in the following paragraphs.

2.2 COPY Command

```
FORMAT:    C source-starting/source-ending/destination-starting

EXAMPLE:   C 8000/8200/1400

DEFAULT
VALUES:    C 1100/1100/1100
```

The COPY command "C" copies the source to destination, which must be RAM. The source is specified by its starting and ending address. The destination is specified by its starting address. To insure that the source is correctly copied, each byte is read after it is written. If a mismatch is detected between source and destination, an error message is printed for each incorrect byte. The message format is similar to that described for the PROGRAM command.

In order to prevent accidental destruction of RAM based programs, the default values for the COPY command are preset to copy the first byte of available program RAM to itself.

2.3 PROGRAM Command

```
        FORMAT:   P source-starting/source-ending/destination-starting

        EXAMPLE:  P 1100/1200/2000

        DEFAULT
        VALUES:   P 1100/18FF/2000
```

The PROGRAM command "P" transfers an NSC Tiny BASIC source program to the MM2716 EPROM (U16 in Figure 1-4). The source program is specified by its starting and ending address. (The ending address of the source may be easily obtained by examining the NSC Tiny BASIC TOP variable.) The source remains unchanged by the programming operation. Since the EPROM programming hardware is mapped into address 2, the starting address of the destination must always begin with hexadecimal "2". The default values for the PROGRAM command fills U16 with the NSC Tiny BASIC program located in the first 2K bytes of available program memory (X'1100 - X'18FF). If a previously programmed EPROM contains a sufficient number of unprogrammed bytes, new programs may be added without erasing the program(s) previously written.

To insure that NSC Tiny BASIC programs are correctly written into EPROM, the PROGRAM command automatically reads each byte after it is written. If a mismatch is detected, the following error message will be printed for each byte:

```
        ADDRESS 8XXX SB XX IS XX
```

The X's above represent hexadecimal digits. The "SB" is an abbreviated notation for "should be". Since the U16 EPROM is mapped into address 2 for READ operations (refer to Figure 1.5), the first digit of the EPROM address will always begin with hexadecimal "8". (The address actually presented on the EPROM address lines is given by the three least significant address digits in the error message.)

2.4 VERIFY Command

FORMAT:	V reference-starting/reference-ending/destination-starting
EXAMPLE:	V 8000/87FF/8800
DEFAULT VALUES:	V 1100/18FF/8000

The VERIFY command "V" verifies the destination against the reference. The reference is specified by its starting and ending address. The destination is specified by its starting address. The reference and destination remain unchanged by the verify operation.

The default values for the VERIFY command cause the U16 EPROM to be verified against the first 2K bytes of available RAM memory (X'1100 - X'18FF).. If a mismatch is detected during verification, an error message will be printed for each incorrect byte. The message format is similar to that described for the PROGRAM command.

The VERIFY command is useful to check the contents of programmed PROMS which may have lost their identification, or may otherwise contain data of doubtful accuracy. It does not need to be used after a "COPY" or a "PROGRAM" command because a verification is performed automatically at the end of each of those functions.

2.5 ERASE CHECK Command

FORMAT:	E source-starting/source-ending/hexadecimal-value
EXAMPLE:	V 1100/11FF/00
DEFAULT VALUES:	V 8000/87FF/FF

The ERASE CHECK command "E" verifies that all bytes contained in the source are equal to the two digit hexadecimal value specified in the last field of the command. The source remains unchanged by the erase check operation.

The "E" command may be used to test whether or not all or part of an MM2716 EPROM is erased. The default values for this command are preset to test that the entire MM2716 EPROM (U16 in Figure 1-4) is erased. If an incorrect byte is located, an error message is printed. The message format is similar to that described for the PROGRAM command.

The "E" command may also be used to locate a specified byte in a given address range. In this case all bytes that are different from the specified hexadecimal value will be flagged as errors.

2.6 FILL Command

```
FORMAT:  F destination-starting/destination-ending/hexadecimal-
           value

EXAMPLE: F 1200/1400/00

DEFAULT
VALUES:  F 1100/18FF/FF
```

The FILL command "F" writes the two digit hexadecimal value specified in the last field of the command to the destination. The destination is specified by its starting and ending address. Since the FILL command reads each byte after it is written, an error message is printed wherever the byte read does not match the byte written. The message format is similar to that described for the PROGRAM command.

The FILL command may be used to fill all or part of available program RAM with the erased value (X'FF) for the MM2716 EPROM. This would normally be done prior to entering a program into RAM. The default values for the FILL command fill the first 2K bytes of available RAM with X'FF. If the FILL command is issued after a program has entered, care should be taken to correctly specify the proper address range or the program may be partially or totally destroyed.

The FILL command may also be used to verify that the program RAM is functioning. This can be accomplished by executing this command several times, using the hexadecimal values X'FF and X'00. This procedure will verify that a logic 0 and a logic 1 can be written to and read from each memory bit.

2.7 DUMP Command

```
FORMAT:  D starting/ending

EXAMPLE: D 8000/80FF

DEFAULT
VALUES:  D 1100/18FF
```

The DUMP command "D" prints out the contents of the specified address range in hexadecimal and ASCII format. Nonprintable ASCII characters are designated by a period. The hexadecimal/ASCII equivalents of sixteen memory bytes are printed out on each line, in the following format:

```
8F00 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDEF
8F10 0D 7F 0A 51 54 59 11 12 2A 2B 2C 54 48 49 53 17 ...QTY..*+, THIS.
```

The four hexadecimal digits at the beginning of each line represent the address of the first (left-most) byte. Memory contents are not affected by the DUMP command. The default values cause the first 2K bytes of available program RAM to be printed out.

2.8 LOAD Command

FORMAT: L displacement

EXAMPLE: L 1ØØØ

DEFAULT VALUES: L Ø

The LOAD command "L" loads an assembly language load module (LM) into memory from a paper tape. (For a detailed description of the LM tape format refer to Appendix E.) The starting memory location where the LM will be stored is specified on the LM tape. If a different starting location is required, an optional displacement (X'ØØØØ - X'FFFF) may be specified in the LOAD command. In this case the starting address will be equal to the address specified on the LM tape plus the displacement specified in the LOAD command. The default value of the displacement is Ø.

The GET routing built into the NSC Tiny BASIC interpreter receives 7-bit ASCII characters which are then stored in memory as 8-bit bytes. These bytes have the most significant bit, B7, set to Ø. Since assembly language LMs require receipt/storage of 8-bit bytes, the GET routine cannot be used to receive assembly language LMs. This problem can be easily overcome by writing an 8-bit GET subroutine in assembly language. This subroutine can then be called, when required, via the NSC Tiny BASIC "LINK" statement. The subroutine requires less than 5Ø bytes and is shown in Appendix D. The bytes that comprise this routine may be entered into RAM, one byte at a time, using the "@" operator.

READ COMMAND

FORMAT: R DISPLACEMENT

EXAMPLE: R 1000

DEFAULT VALUES: R 0

The READ command "R" reads a cassette resident program and stores it into memory. The memory locations at which the program will be stored are specified on the tape as previously described. If it is necessary to read a program into memory at locations other than those specified on the tape, a optional displacement (X'0000 - X'FFFF) may be specified in the "R" command. In this case the starting address for each DATA record will be equal to the address specified on the tape plus the displacement. The default value of the displacement is zero.

If a checksum error is detected when a cassette resident program is read into memory, the user will be alerted by the message "CHECKSUM ERR".

WRITE COMMAND

```
         FORMAT:  W STARTING / ENDING
                    ADDRESS /  ADDRESS

        EXAMPLE:  W 8000/80FF

 DEFAULT VALUES:  W 1100/1EFF
```

The WRITE command "W" writes the contents of the specified memory address range onto audio cassette tape. The memory address range to be written is specified by its starting/ ending addresses. NIBL2 source programs and/or assembly language IM's may be stored on cassette. Stored programs begin with approximately 5 seconds of 0's which serve as leader so that the tape speed has time to stabilize on playback. The leader also serves as an interprogram gap and allows the receiving program to achieve synchronization with the clock pulses.

The tape format consists of one or more DATA records followed by a single END record. A DATA record is organized is as follows:

1) A single character (X'A5) which identifies the start of each record.

2) A byte which specifies the record type. (DATA record=X'01: END record=X'03).

3) A byte which identifies the total number of data bytes in each record. The number of data bytes in a single data record can range from 1 to 256.

4) The least significant byte of the starting address where the data record is to be stored.

5) The most significant byte of the starting address where the data record is to be stored.

6) 1 to 256 data bytes.

7) A single byte checksum (in 2's complement form) of all bytes contained in the data record except for the start of record character, X'A5.

An END record simply consists of the start of record character (X'A5), followed by the record type (X'03) and the 2's complement checksum.

CHAPTER 3

3.1 Loading the EPROM Programming Software Into EPROM

The EPROM programming software shown in Appendix G may be transferred to paper tape so that it can be conveniently used without having to retype it each time it is used. This can be accomplished by initially typing in the program and then turning on the TTY paper tape punch after the LIST command has been entered. This procedure will produce a program listing plus a paper tape version of the program.

Since the EPROM programming software occupies less than 2K bytes, it can be readily programmed into a 2716 EPROM. The resulting EPROM could then be placed into socket U17 in Figure 1-4, so that the EPROM programming software would always be available without having to load a paper tape. If this is done, the contents of one EPROM can still be copied to another using socket U16 only. This can be accomplished by placing the source EPROM into socket U16 and then using the COPY command to transfer the EPROM contents to the first half of RAM (X'1100-X'18FF). Following this, the source EPROM may be removed from socket U16 anD an erased EPROM put in its place. The erased EPROM may then be programmed in the normal manner.

3.2 Loading NSC Tiny BASIC Programs Into RAM

Since the first 2K bytes of available program RAM (X'1000 - X'18FF) are not required by the EPROM programming software, they may be used to store a user's NSC Tiny BASIC program.

Note from Appendix G that the NSC Tiny BASIC variables J, D, M and P all point to scratchpad RAM. The RAM utilized in program memory X'1F00 X'1FFF. (Only a fraction of the bytes in this range are actually used.) If desired, the RAM which is present in the INS8154 may be substituted, making the entire program RAM available for storage of user programs.

3.3 Using the EPROM Programming Software to Program MM2716 EPROMs

The EPROM programming software allows NSC Tiny BASIC programs to be written into EPROM from the keybord, RAM, paper tape, or from another EPROM.

After the user has committed the EPROM programming software to EPROM and placed the latter into socket U17, this software may be executed by entering the following command:

```
>NEW #8800
```

After this is done the "PROGRAM" command, P, may be entered to write the user's RAM resident program into a blank EPROM located in socket U16 shown in Figure 1-4.

Appendices

APPENDIX A

Answers to Exercises

Page 1-21

1. -32767 to 32767, inclusive.

2. Turn switch No. 3 on.

3. 49

4. ERROR 4

5. 2*3 + 4*5 * 6*7 = 6 + 20 + 42 = 68

6. 123*(42/127) = 123 * 0 = 0

7. 16960 The true result, 1,000,000, is larger than 32767.

8. 22/7*1000 = 3*1000 = 3000

9. 1000*22/7 = 22000/7 = 3142

Page 1-25

1. 12 2 35 1

2. 47 9 5 2

3. 26 45

4. 37 73

Page 1-26

5. 100 3218

Page 1-33

1. Simply change Line 60 to read: 60 GO TO 30

Page 1-38

1 (a).

```
        10 PRINT "PROGRAM TO COMPUTE A*X+B"
        15 PRINT ""
        20 PRINT ""
        30 PRINT "A=";
        40 INPUT A
        50 PRINT "B=";
        60 INPUT B
        70 PRINT ""
        80 PRINT "X=";
        90 INPUT X
        100 PRINT "A*X+B  =";
        110 PRINT A*X+B
        120 GO TO 70
```

1 (b).

```
        10 PRINT "PROGRAM TO COMPUTE A*X+B"
        20 PRINT "":PRINT "":PRINT "A=";:INPUT A
        30                   PRINT "B=";:INPUT B
        40 PRINT "";         PRINT "X=";:INPUT X
        50 PRINT "A*X+B  =";:PRINT A*X+B:GO TO 40
```

Line 50 can also be written as follows:

```
        50 PRINT "A*X+B  =", A*X+B:GO TO 40
```

The comma separates the string "A*X+B" and the expression A*X+B.

Page 1-40

Number (Decimal)	Stored As A Byte (Binary)	
3	0000 0011	(2+1)
6	0000 0110	(4+2)
7	0000 0111	(4+2+1)
29	0001 1101	(16+8+4+1)

The largest number that can be represented in a single byte is the "all ones" state:

1111 1111

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

Page 1-48

STATEMENT	N	U	V	W	X
11∅...INPUT N	6844				
12∅ X = MOD (N,16)	6844				12
13∅ N = N/16	427				12
14∅ W = MOD (N,16)	427			11	12
15∅ N = N/16	26			11	12
16∅ V = MOD (N,16)	26		1∅	11	12
17∅ U = N/16	26	1	1∅	11	12

Therefore, NSC Tiny BASIC prints   1 1∅ 11 12

In hexadecimal the number is #1ABC

Page 1-51

```
1∅∅ REM AIR PRESSURE MONITOR AND ALARM
11∅ PRINT " ":PRINT "WHAT IS AIR PRESSURE";:INPUT P
12∅ IF P<13 THEN PRINT "WARNING! AIR PRESSURE TOO HIGH"
13∅ IF P>15 THEN PRINT "WARNING! AIR PRESSURE TOO LOW"
14∅ GO TO 11∅
```

Page 1-61

```
7∅ IF G<>X THEN GO TO 3∅:REM NOT A CORRECT GUESS, GET NEXT
                                 GUESS
```

This replaces both Line 7∅ and Line 8∅ in the program.

The following is an even shorter way to write the program. Try it.

```
1∅ REM GUESS THE NUMBER GAME
2∅ X=RND (1,1∅∅):REM X IS THE SECRET NUMBER FROM 1 TO 1∅∅
3∅ PRINT "":PRINT "WHAT IS YOUR GUESS";
4∅ INPUT G:REM G WILL BE THE GUESS
5∅ IF G<X THEN PRINT "YOUR GUESS IS TOO SMALL":GO TO 3∅
6∅ IF G>X THEN PRINT "YOUR GUESS IS TOO BIG":GO TO 3∅
7∅ PRINT "YOU WIN. LET'S PLAY AGAIN.":GO TO 2∅
```

Page 1-66

1. The results of the RUN will be the same as those shown on page 1-67.

2. No. Try it. See below.

```
>RUN
I   I SQUARED
1   1
2   4
3   9
.   .
.   .
.   .
16  256
.   .
.   .
.   .
```

3. 181

Page 1-68

1. The program will run the same as before.

2. The program will now print values and squares for numbers from 1 to 16.

3. The program will not work. Every line will be 1 1. Furthermore, the program will not stop by itself. You will have to press BREAK.

4. The results will be the same as for the program on page 1-69.

Page 1-72

1.
```
>RUN
1   1
2   3
3   6
4   10
5   15
```

2.
```
>RUN
1   1
2   2
3   6
4   24
5   120
```

3. 15

4. 120

5. 7 28

6.
```
7   5040
8 -25216 Correct answer > 32767
```

Page 1-73

7.
```
>10 PRINT "":PRINT "N=";:INPUT N
 20 S=0
 30 FOR K=1 TO N
 40 S=S+K*K
 50 NEXT K
 60 PRINT "SUMSQUARED =";:PRINT S
```

Line 60 can also be written as follows:

```
60 PRINT "SUMSQUARED =",S
```

Note the comma

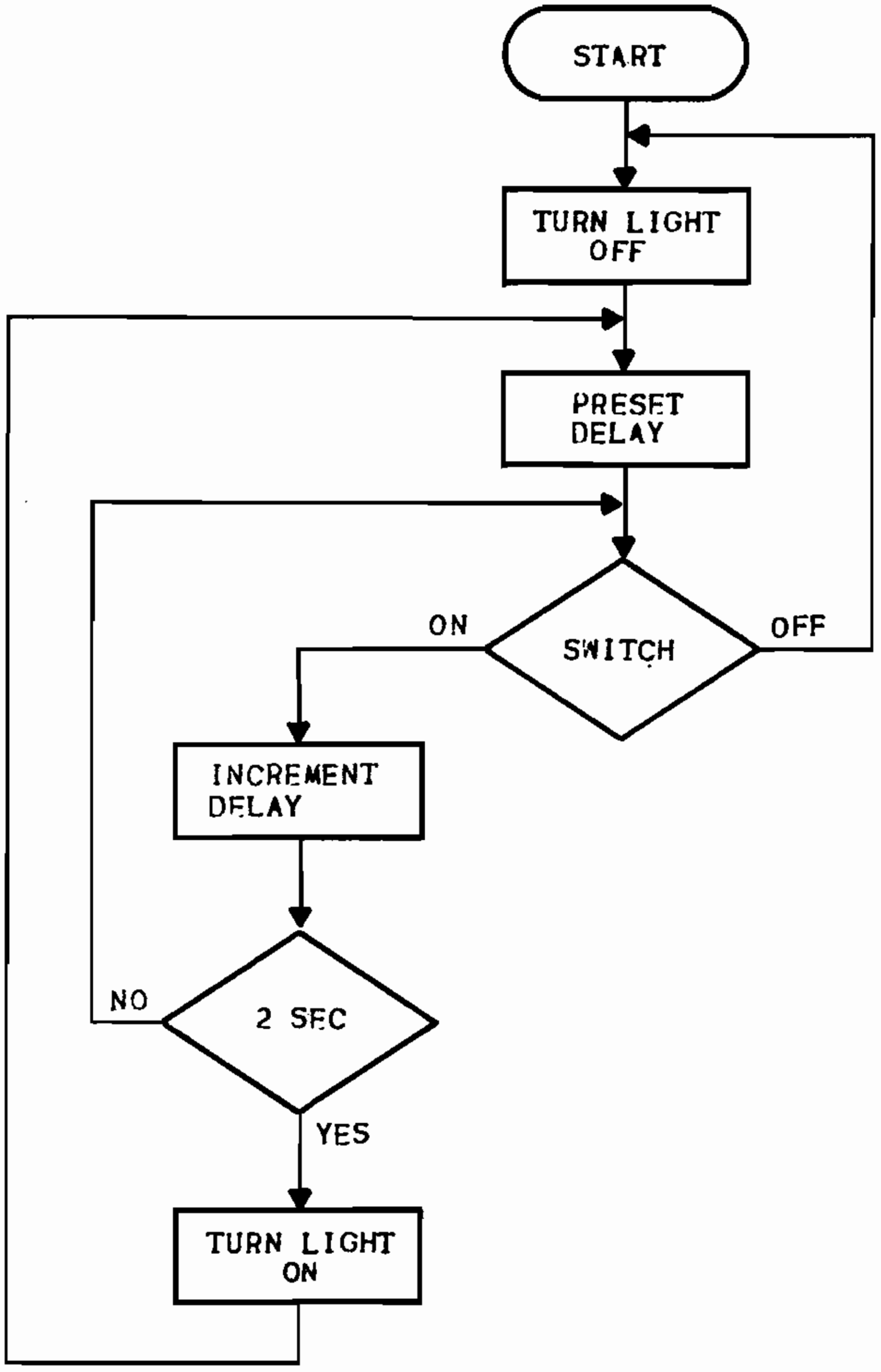
Page 1-94

```
1010 REM STRING COMPARISON SUBROUTINE
1020 REM SET-UP STRING ELEMENT POINTERS, C AND D
1030 C=A:D=B
1040 REM COMPARE PRESENT C & D LOCATIONS. IF UNEQUAL, ERROR
                                                      RETURN
1050 IF@C<>@D PRINT "THE STRINGS ARE UNEQUAL":RETURN
1060 REM IS THIS THE LAST CHARACTER IN THE STRING (CR)?
1070 IF@C=#0D PRINT "THE STRINGS ARE EQUAL":RETURN
1080 REM NONE OF THE ABOVE. CHECK NEXT LOCATION.
1090 C=C+1:D=D+1:GO TO 1050
```

Page 1-102

```
10 A=#EFFF
20 @A=1:REM TURN LIGHT OFF
30 B=0
40 DO
50 IF @A=1 THEN GO TO 20:SWITCH IS OFF
60 B=B+1
70 DELAY 10
80 UNTIL B=200:REM SWITCH MUSE BE CLOSED 2 SEC
90 @A=0:GO TO 30:REM TURN LIGHT ON
```

Flowchart


START
TURN LIGHT OFF
PRESET DELAY
SWITCH
ON
OFF
INCREMENT DELAY
2 SEC
NO
YES
TURN LIGHT ON

APPENDIX B

Error Code Summary

Error Number	Explanation
1	Out of memory
2	Statement used improperly
3	Unexpected character (after legal statement)
4	Syntax error
5	Value (format) error
6	Ending quote missing from string
7	GO target line does not exist
8	RETURN without previous GOSUB
9	Expression, FOR-NEXT, DO-UNTIL or GOSUB nested too deeply
10	NEXT without previous matching FOR
11	UNTIL without previous DO
12	Division by zero

APPENDIX C

ASCII Codes

The following table contains the 7-bit hexadecimal code for each character in the ASCII character set.

ASCII Character Set in Hexadecimal Representation

7-bit Hexa-decimal Number	Char-acter	7-bit Hexa-decimal Number	Char-acter	7-bit Hexa-decimal Number	Char-acter	7-bit Hexa-decimal Number	Char-acter
ØØ	NUL	2Ø	SP	4Ø	@	6Ø	
Ø1	SOH	21	!	41	A	61	a
Ø2	STX	22	"	42	B	62	b
Ø3	ETX	23	#	43	C	63	c
Ø4	EOT	24	$	44	D	64	d
Ø5	ENQ	25	%	45	E	65	e
Ø6	ACK	26	&	46	F	66	f
Ø7	BEL	27	'	47	G	67	g
Ø8	BS	28	(	48	H	68	h
Ø9	HT	29	)	49	I	69	i
ØA	LF	2A	*	4A	J	6A	j
ØB	VT	2B	+	4B	K	6B	k
ØC	FF	2C	,	4C	L	6C	l
ØD	CR	2D	–	4D	M	6D	m
ØE	SO	2E	.	4E	N	6E	n
ØF	SI	2F	/	4F	O	6F	o
1Ø	DLE	3Ø	Ø	5Ø	P	7Ø	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	
1C	FS	3C	<	5C	\	7C	
1D	GS	3D	=	5D	]	7D	ALT
1E	RS	3E	>	5E		7E	ESC
1F	US	3F	?	5F		7F	DEL, RUB

Definitions of Non-printing Characters

Character	Definition
NUL	Null
SOH	Start of Heading (also start of message)
STX	Start of Text (also EOA-end of address)
ETX	End of Text (also EOM-end of message)
EOT	End of Transmission (aalso END)
ENQ	Enquiry (also ENQRY, WRU)
ACK	Acknowledge (also RU)
BEL	Bell
BS	Backspace
HT	Horizontal Tab
LF	Line Feed
VT	Vertical Tab (VTAB)
FF	Form Feed
CR	Carriage Return
SO	Shift Out
SI	Shift In
DLE	Data Link Escape
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4
NAK	Negative Acknowledge
SYN	Synchronous Idle
ETB	End of Transmission Block
CAN	Cancel (CANCL)
EM	End of Medium
SUB	Substitute
ESC	Escape
FS	File Separator
GS	Group Separator
RS	Record Separator
US	Unit Separator
SP	Space
ALT	Alt Mode
ESC	Escape
DEL, RUB	Delete or Rubout

APPENDIX D

NSC Tiny BASIC Language Summary

STATEMENTS (except for INPUT, may be used as commands)

NEW expr	Establishes a new start-of-program address equal to the value of 'expr'. NSC Tiny BASIC then executes its initialization sequence which clears all variables, resets all hardware/software stacks, disables interrupts, enables BREAK capability from the console, and performs the nondestructive RAM search described in Chapter 2, Section 2. If the value of 'expr' points to a ROM address, the NSC Tiny BASIC program which begins at this address will be automatically executed. Program memory (including the end-of-program pointer used by the editor) is not altered by this command.
NEW	Sets the end-of-program pointer equal to the start-of-program pointer so that a new program may be entered. If a program already exists at the start-of-program address, it will be lost.
RUN	Runs the current program.
CONT	Continues execution of the current program from the point where execution was suspended (via a STOP, console interrupt, or reset).
LIST (expr)	Lists the current program (optionally starting at the line number specified by (expr).
REM anything	Remark (no operation).
CLEAR	Initializes all variables to Ø, disables interrupts, enables BREAK capability from the console, and resets all stacks (GOSUB, FOR-NEXT, DO-UNTIL).
[LET] var = expr	Assigns expression value to variable.
[LET] STAT = expr	Sets the STATUS word equal to the least significant byte of 'expr'. When the STATUS word is used to enable interrupts at the hardware, processing will be deferred for one statement.
[LET] @factor = expr	Sets the memory location pointed to by 'factor' equal to the least significant byte of 'expr'.

[LET] $factor = "string"	Assigns a string in RAM starting at the address 'factor'. Strings are terminated by a carriage return.
[LET] factor = factor	Memory to memory string assignment,(copy).
PRINT expr	Prints the value of 'expr'.
PRINT "string"	Prints the string.
PRINT $factor	Prints the string starting at the memory address 'factor'.
IF expr [THEN] statement(s)	Remainder of the program line is executed if expr is true (non-zero).
FOR var = expr TO expr [STEP expr]	FOR loop initialization. FOR loops may be nested up to four levels deep.
NEXT var	FOR loop termination.
DO	DO loop initiation. DO loops may be nested up to eight levels deep.
UNTIL expr	DO loop termination.
GO TO expr	Transfer control to statement number 'expr'.
GOSUB expr	Call subroutine at statement number 'expr'. Subroutine (including those servicing interrupts) may be nested up to eight levels deep.
RETURN	Return from subroutine.
INPUT var	Read value from console into variable.
INPUT $factor	Read string from console into memory beginning at address 'factor'.
LINK expr	Links to an assembly language subroutine which begins at the address 'expr'.
ON expr1, expr2	Interrupt processing definition. When interrupt number expr1 occurs, NSC Tiny BASIC will execute a GOSUB beginning at line number expr2. If expr 2 is zero, the corresponding interrupt is disabled at the software level. Interrupt numbers may be 1 or 2. Use of the ON statement disables console interrupts (BREAK function). Interrupts must also be enabled at the hardware level by setting the Interrupt Enable bit in the status register (using STAT=1, for example).

DELAY expr	Delay for expr time units (nominally milliseconds, 1-1040). Delay 0 gives the maximum delay of 1040 milliseconds.
STOP	Terminate program execution. A message is printed and NSC Tiny BASIC returns to COMMAND mode.

OPERATORS

Arithmetic operators:	addition	+
	subtraction	-
	multiplication	*
	division	/
Relational operators:	less than	<
	greater than	>
	equal to	=
	no equal to	<>
	less than or equal to	<=
	greater than or equal to	>=
Logical operators:	logical AND	AND
	logical OR	OR
	logical NOT	NOT

@factor	Read a byte from memory/peripheral, or write a byte to memory/peripheral. Factor is the memory/peripheral address.

FUNCTIONS

STAT	Status Register contents.
TOP	Top-Of-Program address (first available memory address after end-of-program byte).
INC (X), DEC (X)	Increment or decrement a memory location (non-interruptable for multiprocessing).
MOD (X,Y)	Modulus function (remainder of x/y).
RND (X,Y)	Random number generator (in interval x,y).

COMMANDS (cannot be used as statements)

NEW expr	Establishes a new start-of-program address equal to the value of expr. NSC Tiny BASIC then executes its initialization sequence which clears all variables, resets all hardware/software stacks, disables interrupts, enables BREAK capability from the console, and performs the non-destructive RAM search described in Section II. If the value of expr points to a ROM address, the NSC Tiny BASIC program which begins at this address will be automatically executed and program memory (including the end-of-program pointer used by the editor) is not altered by this command.
NEW	NEW followed only by a carriage return sets the end-of-program pointer equal to the start-of-program pointer so that a new program may be entered. If a program already exists at the start-of-program address, it will be lost.
RUN	Runs the current program.
CONT	Continues execution of the current program from the point where execution was suspended (via a STOP, console interrupt or reset).
LIST expr	Lists the current program (optionally starting at the line number specified by expr).

APPENDIX E

NSC Tiny BASIC Grammar

All items in single quotes are actual symbols in NSC Tiny BASIC; all other identifiers are symbols in 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 occurrences; and the semicolon, ";", marks the end of a definition.

```
NSC Tiny BASIC - line = Immediate-statement
                      ! Program-line
                      ;

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

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

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

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-list
          ! 'DO'
          ! 'UNTIL' Rel-exp
          ! 'FOR' Variable '=' Rel-exp 'TO' Rel-exp ('STEP' Rel-exp)?
          ! 'NEXT' Variable
          ! 'INPUT' '$' Factor ! Variable-list
          ! 'LINK' Rel-exp
          ! 'REM' Any-Character-Except-Carriage-Return *
          ! 'STOP'
          ! 'CLEAR'
          ! 'DELAY' Rel-exp
          ! 'ON' Rel-exp ',' ? Rel-exp
          ;
```

```
Factor = (Variable ! Number ! Function ! '(' Rel-exp ')');
Left-part = (Variable ! '@' Factor ! 'STAT') ;
REL-EXP = Term rel-op term
STRING = '"' Any-character-except-"-or-CR'"'
VARIABLE = 'A' ! 'B' !......'Z'
VARIABLE-LIST = Variable (',' Variable)*
PRINT-ITEM = (Rel-exp ! '$'factor ! String)
PRINT-LIST = PRINT-ITEM (',' PRINT-ITEM)*(';')?
Function = 'MOD' '(' Rel-exp ',' Rel-exp ')'
         ! 'RND' '(' Rel-exp ',' Rel-exp ')'
         ! 'INC' '(' Rel-exp ')'
         ! 'DEC' '(' Rel-exp ')'
         ;
Term = Factor Termop Factor
```

APPENDIX F

```
1J=7936:$J="0123456789ABCDEF":D=J+17:M=D+8:P=M+20:PR"CMD";
2INPUT$P:X=4352:Y=6399:Z=#FF:G=27:GOSUB50:C=T:GOSUB50:F=T:P=P-1:[illegible]
3IFC=67Y=4352:Z=Y:GOTO15
4IFC=86Z=#8000:GOTO15
5IFC=69X=#8000:Y=#87FF:G=31:GOTO15
6IFC=70G=31:GOTO15
7IFC=80Z=8192:G=35:GOTO15
8IFC=76X=0:Y=1:Z=0:S=1:G=38:GOTO15
9IFC=65X=0:Y=0:Z=0:S=1:G=39:GOTO15
10IFC=68S=2:G=42:GOTO15
11IFC=87Y=8191:S=2:G=46:GOTO15
12IFC=82X=0:Z=0:S=1:G=47:GOTO15
13PR"INPUT ERR":GOTO1
14PR"DONE":GOTO1
15N=1:IFF=#DGOTOG
16R=0
17GOSUB50:IF(T=#D)OR(T=47)OR(T=44)GOTO21
18T=T-48:IF(T<0)OR(T>22)OR((T>9)AND(T<17))GOTO13
19IFT>9T=T-7
20R=16*R+T:GOTO17
21IFN=1X=R:GOTO24
22IFN=2Y=R:GOTO24
23Z=R
24N=N+1:IFN>3GOTOG
25IFT=#DGOTO13
26GOTO16
27FORI=XTOY:IFC=67@Z=@I
28IF@Z=@IGOTO30
29A=Z:B=@I:C=@Z:GOSUB48
30Z=Z+1:NEXTI:GOTO14
31FORI=XTOY:IFC=70@I=Z
32IF@I=ZGOTO34
33A=I:B=Z:C=@I:GOSUB48
34NEXTI:GOTO14
35FORI=0TO(Y-X):@(Z+I)=@(X+I):IF@(#6000+Z+I)=@(X+I)GOTO37
36A=#6000+Z+I:B=@(X+I):C=@(#6000+Z+I):GOSUB48
37NEXTI:GOTO14
38PR"TURN READER ON":LINK#8D81:GOTO40
39PR"CONNECT RS-232":LINK#8D88
40IFZ=1PR"CKSUM ERR":GOTO1
41GOTO14
42FORI=XTOYSTEP16:H=I:LINK#8E4E:$M="................"
43FORL=ITOI+15:H=@L:LINK#8E3F
44IF(H>31)AND(H<#7B)@(M+L-I)=H
45NEXTL:PR$M:NEXTI:GOTO14
46LINK#8E8F:GOTO14
47LINK#8F65:GOTO40
48PR"ADDRESS ";:H=A:LINK#8E4E:PR"SB ";:H=B:LINK#8E3F:PR"IS ";:H=C
49LINK#8E3F:PR"":RETURN
50T=@P:P=P+1:IFT=32GOTO50
51RETURN
```

APPENDIX G

```
   REM *****  8070 UTILITY PROGRAM  *****
   REM ***** BY RON PASQUALINI, NSC *****

   REM This version of the program has been expanded for
   REM legibility.  It will not run as shown since some
   REM lines do not have line numbers.  The compressed
   REM version of the program is functionally identical
   REM and will run properly.


   REM Initialize variables and Prompt for command

 1 J=7936 : $J="0123456789ABCDEF"
   D=J+17 : M=D+3 : P=J+20
   PR "CMD":
 2 INPUT $P
   X=4352 : Y=6399 : Z=#FF : G=27 : GOSUB 50
   C=T : GOSUB 50 : F=T : P=P-1 : S=3

   REM Test command letter, setting default values
   REM as needed for the entered command.

 3 IF C=67 Y=4352 : Z=Y : GOTO 15                     : REM "C"
 4 IF C=86 Z=#8000 : GOTO 15                          : REM "V"
 5 IF C=69 X=#8000 : Y=#87FF : G=31 : GOTO 15         : REM "E"
 6 IF C=70 G=31 : GOTO 15                             : REM "F"
 7 IF C=80 Z=8192 : G=35 : GOTO 15                    : REM "P"
 8 IF C=76 X=0 : Y=1 : Z=0 : S=1 : G=38 : GOTO 15 : REM "L"
 9 IF C=65 X=0 : Y=0 : Z=0 : S=1 : G=39 : GOTO 15 : REM "A"
10 IF C=68 S=2 : G=42 : GOTO 15                       : REM "D"
11 IF C=87 Y=8191 : S=2 : G=46 : GOTO 15              : REM "W"
12 IF C=82 X=0 : Z=0 : S=1 : G=47 : GOTO 15          : REM "R"

   REM Error messages

13 PR "INPUT ERR" : GOTO 1
14 PR "DONE" : GOTO 1

   REM Process command operands

15 N=1 : IF F=#D GOTO G
16 R=0
17 GOSUB 50 : IF (T=#D) OR (T=47) OR (T=44) GOTO 21
18 T=T-48 : IF (T<0) OR (T>22) OR ((T>9) AND (T<17)) GOTO 13
19 IF T>9 T=T-7
20 R=16*R+T : GOTO 17
21 IF N=1 X=R : GOTO 24
22 IF N=2 Y=R : GOTO 24
23 Z=R
24 N=N+1 : IF N>S GOTO G
25 IF T=#D GOTO 13
26 GOTO 16

   REM Process "C", "V" Commands

27 FOR I=X TO Y
     IF C=67 @Z=@I
28   IF @Z=@I GOTO 30
```

```
29    A=Z : B=@I : C=@Z : GOSUB 48
30    Z=Z+1
   NEXT I
   GOTO 14

   REM Process "E", "F" Commands

31 FOR I=X TO Y
     IF C=70 @I=Z
32   IF @I=Z GOTO 34
33   A=I : B=Z : C=@I : GOSUB 48
34 NEXT I
   GOTO 14

   REM Process "P" Command

35 FOR I=0 TO (Y-X)
     @(Z+I)=@(X+I)
     IF @(#6000+Z+I)=@(X+I) GOTO 37
36   A=#6000+Z+I : B=@(X+I)
     C=@( #6000+Z+I) : GOSUB 48
37 NEXT I
   GOTO 14

   REM Process "L" Command

38 PR "TURN READER ON" : LINK #8D81 : GOTO 40

   REM Process "A" Command

39 PR "CONNECT RS-232" : LINK #8D88

   REM Post processing for "L", "A" Commands

40 IF Z=1 PR "CKSUM ERR" : GOTO 1
41 GOTO 14

   REM Process "D" Command

42 FOR I=X TO Y STEP 16
     H=I : LINK #8E4E : $M="................"
43   FOR L=I TO I+15
       H=@L : LINK #8E3F
44     IF (H>31) AND (H<#7B) @(M+L-I)=H
45   NEXT L
     PR $M
   NEXT I
   GOTO 14

   REM Process "W" Command

46 LINK #8E8F : GOTO 14

   REM Process "R" Command

47 LINK #8F65 : GOTO 40

   REM Subroutine: Print verification error
```

```
48 PR "ADDRESS "; : H=A : LINK #8E4E
   PR "SR "; : H=B : LINK #8E3F
   PR "IS "; : H=C
49 LINK #8E3F : PR "" : RETURN

   REM Subroutine: Get next character from input buffer

50 T=@P : P=P+1 : IF T=32 GOTO 50
51 RETURN
```

APPENDIX H

```
            .TITLE  AS8070,' 8070 UTILITY SUBROUTINES'
     2F     .LIST   02F



        ;                    'ASSYLANG' PROGRAM
        ;                           BY
        ;                     RON PASQUALINI
        ;                 NATIONAL SEMICONDUCTOR

        ; 'ASSYLANG' INCLUDES  ALL  OF  THE  ASSEMBLY  LANGUAGE
        ; SUBROUTINES  REQUIRED BY THE NIBL? PROGRAM 'UTILITY'.



 0000       .=08D81

            .INCLD ASCILD
```

```
                              .PAGE    'ASCILD'

                    ; 'ASCILD' DOWNLOADS AN ASSEMBLY LANGUAGE LM OR NIBL2  SOURCE
                    ; PROGRAM AT 4800 BAUD IN RESPONSE TO THE COMMAND 'A'.  IT IS
                    ; ALSO  ENTERED  TO  LOAD  AN ASSEMBLY LANGUAGE LM FROM PAPER
                    ; TAPE AT 110 BAUD IN RESPONSE TO THE COMMAND 'L'.  FOR  BOTH
                    ; COMMANDS  THE  DOWNLOADED  PROGRAM  MUST  BE IN THE FORM OF
                    ; ASCII  CHARACTERS.  THIS  SUBROUTINE  CALLS  THE  NIBL2
                    ; SUBROUTINE 'GECO'.

                    ; TWO  ASCII  CHARACTERS  EQUAL  ONE  BYTE,  WITH  THE  MOST
                    ; SIGNIFICANT  NIBBLE  (MSN)  LOCATED  AT  THE  LOWER  MEMORY
                    ; ADDRESS.  THE LEAST SIGNIFICANT NIBBLE (LSN) IS LOCATED  AT
                    ; THE HIGHER MEMORY ADDRESS.

                    ; THE FILE FORMAT FOR A DATA RECORD IS AS FOLLOWS:
                    ;
                    ;   1) START OF RECORD CHARACTER (X'02)
                    ;   2) RECORD LENGTH, X'01-X'FF, (2 ASCII CHAR)
                    ;   3) MSB OF RECORD LOAD ADDRESS (2 ASCII CHAR)
                    ;   4) LSB OF RECORD LOAD ADDRESS (2 ASCII CHAR)
                    ;   5) RECORD TYPE (2 ASCII CHAR)
                    ;      (DATA RECORD=X'00, END RECORD=X'01)
                    ;   6) 1-255 DATA BYTES (2 ASCII CHAR PER BYTE)
                    ;   7) RECORD CHECKSUM OF THE HEXADECIMAL
                    ;      EQUIVALENT OF ALL BYTES IN 2) THRU 6)
                    ;      IN 2'S COMPLEMENT FORM

                    ; THE BAUD RATE FOR DOWNLOADING CAN BE 110 OR OR  4800  BAUD,
                    ; DEPENDING UPON THE SUBROUTINE ENTRY POINT.

                    ; THE 'ASCILD' SUBROUTINE REQUIRES 8 BYTES OF STACK MEMORY  &
                    ; DESTROYS  A,  E,  P2  & P3.  THE NIBL2 SUBROUTINE 'GECO' IS
                    ; CALLED.

                    ; DISPLACEMENTS RELATIVE TO P3:

     0000                NBYTES   = 0            ; NUMBER OF DATA BYTES IN
                                                 ; DATA RECORD
     0001                MCKSUM   = 1            ; CHECKSUM FORMED IN MEMORY
     0002                RECTYP   = 2            ; RECORD TYPE
     0003                MSN      = 3            ; MOST SIGNIFICANT NIBBLE
                                                 ; OF 8 BIT BYTE IN 'GETBYT'
                                                 ; SUBROUTINE
     0004                CTR      = 4            ; CTR FOR DELAY @ END OF SUBR
     0009                BYTCTR   = 9            ; BYTE CTR FOR 'HEX2AS' SUBR


                    ; DISPLACEMENTS RELATIVE TO P2:

     0006                .SET     D,6            ; CONTENTS OF NIBL2 VARIABLE
                                                 ; D POINTS TO MEMORY LOCATION
                                                 ; WHERE ASCII EQUIV OF 2/4
                                                 ; DIGIT HEX NUMBER IS STORED.
```

```
                                                  ; MS CHAR IS AT LOWEST MEMORY
                                                  ; ADDRESS, TO WHICH D POINTS
         000E            .SET     H,14            ; CONTENTS OF NIBL2 VARIABLE
                                                  ; H EQUAL STARTING ADDRESS OF
                                                  ; EACH DATA RECORD
         002E            .SET     X,46            ; DISPLACEMENT FOR RECORD
                                                  ; STARTING ADDRESS
                                                  ; (NIBL2 VARIABLE X)
         0030            .SET     Y,48            ; BAUD RATE INDICATOR FLAG
                                                  ; WHICH ALLOWS PRINTOUT OF
                                                  ; STARTING ADDRESS OF EACH
                                                  ; RECORD AT 110 BAUD ONLY.
                                                  ; (NIBL2 VARIABLE Y)
         0032            .SET     Z,50            ; CHECKSUM ERROR FLAG
                                                  ; (NIBL2 VARIABLE Z)


         FF00            .SET     SRAM,0FF00      ; STARTING LOCATION
                                                  ; OF SCRATCH RAM TO
                                                  ; WHICH P3 POINTS


         FFEC            FDELAY   = 0FFEC         ; LOCATION WHERE DLY CONST
                                                  ; FOR 1 BIT DLY IS STORED
                                                  ; (REQUIRED BY NIBL2 'GECO'
                                                  ; SUBROUTINE)
         0933            GECO     = 00933         ; ENTRY POINT FOR
                                                  ; NIBL2 'GECO' SUBROUTINE


                ASCILD:                           ; BAUD RATE =110
    8D81 845202 B110:    LD       EA,=X'252       ; STORE DELAY COUNT FOR 1 BIT
    8D84 8DEC            ST       EA,FDELAY       ; DELAY @ 110 BAUD

    8D86 7405            BRA      NEXT            ; CONTINUE

                                                  ; BAUD RATE =4800
    8D88 840400 B4800:   LD       EA,=04          ; STORE DELAY COUNT FOR 1 BIT
    8D8B 8DEC            ST       EA,FDELAY       ; DELAY @ 4800 BAUD

    8D8D 39FE   NEXT:    AND      S,=0FE          ; DISABLE INTERRUPTS

    8D8F 2700FF          LD       P3,=SRAM        ; POINT P3 TO SCRATCH RAM

    8D92 203209 LOOP1:   JSR      GECO            ; ASCII CHAR-->A REG & E REG

    8D95 06     NSTOP1:  LD       A,S             ; SAMPLE SA & WAIT UNTIL
    8D96 D410            AND      A,=010          ; SA=1. (PARITY BIT=1 OR
    8D98 6CFB            BZ       NSTOP1          ; STOP BIT=1 HAS OCCURRED)

                                                  ; SA=1
    8D9A 40              LD       A,E             ; ASCII CHAR-->A REG

    8D9B E402            XOR      A,=02           ; LOOP IF CHAR <> X'02
```

```
8D9D 70F3         BNZ     LOOP1          ; (START OF RECORD)

                                         ; X'02 FOUND
8D9F 20168E       JSR     GETBYT         ; GET RECORD LENGTH (IN HEX
8DA2 CB00         ST      A,NBYTES,P3    ; AND SAVE IT

8DA4 CB01         ST      A,MCKSUM,P3    ; INITIALIZE MEMORY CHECKSUM

8DA6 20168E       JSR     GETBYT         ; GET MSB OF LOAD ADDRESS
8DA9 0A           PUSH    A              ; (ADDRH) & SAVE ON THE STACK

8DAA F301         ADD     A,MCKSUM,P3    ; UPDATE MEMORY CHECKSUM
8DAC CB01         ST      A,MCKSUM,P3

8DAE 20168E       JSR     GETBYT         ; GET LSB OF LOAD ADDRESS
                                         ; (ADDRL) IN A REG & E REG

8DB1 F301         ADD     A,MCKSUM,P3    ; UPDATE MEMORY CHECKSUM
8DB3 CB01         ST      A,MCKSUM,P3

8DB5 38           POP     A              ; ADDRH-->A REG
8DB6 01           XCH     A,E            ; ADDRH-->E REG, ADDRL-->A REG

8DB7 260010       LD      P2,=01000      ; POINT P2 TO NIBL2 VARIABLES
8DBA B22E         ADD     EA,X,P2        ; ADD DISPL (NIBL2 VAR X)

8DBC 8A0E         ST      EA,H,P2        ; SAVE RECORD STARTING ADDR
                                         ; IN NIBL2 VARIABLE H

8DBE 20168E       JSR     GETBYT         ; GET RECORD TYPE IN A REG
8DC1 CB02         ST      A,RECTYP,P3    ; & E REG & SAVE IT

8DC3 F301         ADD     A,MCKSUM,P3    ; UPDATE MEMORY CHECKSUM
8DC5 CB01         ST      A,MCKSUM,P3

8DC7 40           LD      A,E            ; RECORD TYPE-->A REG

8DC8 7C20         BNZ     NOTDTA         ; IF RECORD TYPE = 0
                                         ; IT IS A DATA RECORD

                                         ; A REG=0, RECORD IS
                                         ; A DATA RECORD

                                         ; IF BAUD RATE=110 PRINT
                                         ; OUT RECORD STARTING
                                         ; ADDRESS

8DCA 8230         LD      EA,Y,P2        ; IF Y=1 BAUD RATE=110
8DCC E401         XOR     A,=01
8DCE 7C0A         BNZ     DTAREC

                                         ; BAUD RATE=110
                                         ; PRINT OUT RECORD
                                         ; STARTING ADDRESS
```

```
8DD0 18                CALL    8              ; PRINT CR/LF TO PUT RECORD
                                              ; ADDRESS ON A NEW LINE.
                                              ; (NECESSARY BECAUSE 'GECO',
                                              ; NOT 'GETC', MUST BE USED).

8DD1 22118E            PLI     P2,=ADRMSG     ; POINT P2 TO 1ST CHAR OF
                                              ; ADDR MSG & SAVE OLD P2
8DD4 1E                CALL    14             ; PRINT OUT THE MESSAGE:
                                              ; "ADDR=X'" WITHOUT CR/LF

8DD5 204D8E            JSR     PRT4           ; PRINT OUT 4 CHAR ASCII
                                              ; EQUIV OF 16 BIT HEX
                                              ; STARTING ADDRESS
                                              ; WITHOUT CR/LF

8DD8 18                CALL    8              ; PRINT CR/LF

8DD9 5E                POP     P2             ; RESTORE OLD P2

8DDA 820E    DTAREC:   LD      EA,H,P2        ; RECORD START ADDR-->EA REG
8DDC 46                LD      P2,EA          ; RECORD START ADDR-->P2

8DDD 20168E NXTBYT:    JSR     GETBYT         ; GET DATA BYTE

8DE0 CE01              ST      A,@+1,P2       ; STORE BYTE & INCR PTR

8DE2 F301              ADD     A,MCKSUM,P3    ; UPDATE MEMORY CHECKSUM
8DE4 CB01              ST      A,MCKSUM,P3

8DE6 9B00              DLD     A,NBYTES,P3    ; DECREMENT BYTE COUNT
8DE8 7CF3              BNZ     NXTBYT         ; & LOOP IF COUNT <> 0

8DEA 20168E NOTDTA:    JSR     GETBYT         ; GET RECEIVED CHECKSUM

8DED 0A                PUSH    A              ; SAVE CHECKSUM ON STACK
8DEE 18                CALL    8              ; PRINT CR/LF
8DEF 38                POP     A              ; RESTORE CHECKSUM TO A REG

8DF0 F301              ADD     A,MCKSUM,P3    ; ADD MEMORY CHECKSUM

8DF2 6C08              BZ      RECTST         ; TEST SUM FOR ZERO

                                              ; SUM <> 0. CHECKSUM
                                              ; ERROR HAS OCCURRED
8DF4 260010            LD      P2,=01000      ; POINT P2 TO NIBL2 VARIABLES
8DF7 840100            LD      EA,=01         ; SET NIBL2 VARIABLE Z=1
8DFA 8A32              ST      EA,Z,P2        ; TO INDICATE CHECKSUM ERROR

8DFC C302    RECTST:   LD      A,RECTYP,P3    ; LOAD RECORD TYPE & TEST
8DFE E401              XOR     A,=01          ; FOR END RECORD = X'01

8E00 7C90              BNZ     LOOP1          ; GET NEXT RECORD IF CURRENT
                                              ; RECORD IS NOT AN END RECORD
```

```
                                                ; CURRENT RECORD = END RECORD.
                                                ; DELAY APPROX .54 SEC IN ORDER
                                                ; TO SLEW OFF NULLS AT END OF
                                                ; END RECORD.  THIS WILL INSURE
                                                ; THAT SA=1 UPON RETURN TO NIBL2
                                                ; PROGRAM @ 4800 BAUD.  THIS
                                                ; DELAY IS ONLY REQUIRED AT 4800
                                                ; BAUD, BUT DOES NOT HAVE ANY
                                                ; DETRIMENTAL EFFECT AT 110
                                                ; BAUD, USING A TTY WITH A
                                                ; READER RELAY.
8F02 C496              LD       A,=150          ; SAVE # OF TIMES 'DELAY'
8E04 CB04              ST       A,CTR,P3        ; WILL BE CALLED

8F06 C4FF      L1:     LD       A,=0FF          ; LOAD DELAY COUNT
8F08 20F78F            JSR      DELAY           ; DELAY FOR 3609 USEC

8F0B 9B04              DLD      A,CTR,P3        ; DECREMENT & LOAD LOOP COUNT
8F0D 7CF7              BNZ      L1              ; & REPEAT IF LOOP COUNT <> 0

8F0F 18                CALL     8               ; PRINT CR/LF @ END OF RECORD
                                                ; (FOR 110 BAUD USING TTY)

8F10 5C                RET                      ; RETURN


8F11 4144      ADRMSG: .ASCII   'ADDR='         ; ADDRESS MSG FOR 110 BAUD
8E16 A7                .BYTE    '''' + 080
```

```
                         .PAGE   'ASCILD - GETBYT'

                  ; 'GETBYT' GETS TWO ASCII CHARACTERS AND COMBINES THEM INTO A
                    SINGLE 8 BIT BYTE.

                  ; 'GETBYT' REQUIRES 1 BYTE OF SCRATCH RAM (MSN), USES 4 BYTES
                  ; OF STACK MEMORY, CALLS THE  NIBL2  SUBROUTINE  'GECO',  AND
                  ; DESTROYS REGISTER A AND REGISTER E.

                  ; THE BYTE WHICH IS GOTTEN IS RETURNED IN THE A REGISTER  AND
                  ; THE E REGISTER.

                  ; SUBROUTINE PARAMETERS INCLUDE:

                          ;       MSN         DISPLACEMENT RELATIVE TO P3


8E17 203209 GETBYT: JSR     GECO          ; GET ASCII EQUIV OF MSN
                                          ; IN A REG & E REG

8E1A 06     NSTOP2: LD      A,S           ; SAMPLE SA & WAIT UNTIL
8E1B D410           AND     A,=010        ; SA=1. (PARITY BIT=1 OR
8E1D 6CFB           BZ      NSTOP2        ; STOP BIT=1 HAS OCCURRED)

                                          ; SA=1
8E1F 40             LD      A,E           ; ASCII EQUIV OF MSN-->A REG

8E20 2D02           BND     ATOF1         ; TEST FOR X'30<=MSN<=X'39

8E22 7402           BRA     SHIFT1        ; X'00<=A REG<=X'09. CONTINUE

8E24 FC37   ATOF1:  SUB     A,=X'37       ; CONVERT ASCII A THRU F
                                          ; TO HEX A THRU F

8E26 0E     SHIFT1: SL      A             ; SHIFT MSN 4 BITS
8E27 0E             SL      A             ; TO THE LEFT, PLACING
8E28 0E             SL      A             ; ZEROS INTO LSN
8E29 0E             SL      A             ; POSITION

8E2A CB03           ST      A,MSN,P3      ; SAVE MSN

8E2C 203209         JSR     GECO          ; GET ASCII EQUIV OF LSN
                                          ; IN A REG & E REG

8E2F 06     NSTOP3: LD      A,S           ; SAMPLE SA & WAIT UNTIL
8E30 D410           AND     A,=010        ; SA=1. (PARITY BIT=1 OR
8E32 6CFB           BZ      NSTOP3        ; STOP BIT=1 HAS OCCURRED)

                                          ; SA=1
8E34 40             LD      A,E           ; ASCII EQUIV OF LSN-->A REG

8E35 2D02           BND     ATOF2         ; TEST FOR X'30<=LSN<=X'39

8E37 7402           BRA     ORNIBL        ; X'00<=A REG<=X'09. CONTINUE
```

```
    8E39 FC37   ATOF2:  SUB     A,=X'37      ; CONVERT ASCII A THRU F
                                             ; TO HEX A THRU F

    8E3B DB03   ORNIBL: OR      A,MSN,P3     ; OR MSN WITH LSN TO
                                             ; FORM 8 BIT CHARACTER

    8E3D 48             LD      E,A          ; PUT CHAR INTO E REG

    8E3E 5C             RET                  ; RETURN
  19                    .INCLD HEX2ASCI
```

```
                                .PAGE    'HEX2ASCI'

                        ; 'HEX2ASCI'  WILL  CONVERT  A  HEX  NUMBER  TO   ITS   ASCII
                        ; EQUIVALENT  AND  PRINT  THE  NUMBER  WITH  A TRAILING SPACE
                        ; APPENDED.  2 DIGIT & 4 DIGIT HEX NUMBERS MAY  BE  CONVERTED
                        ; DEPENDING UPON THE ENTRY POINT.

                        ; THE SUBROUTINE REQUIRES 6 BYTES OF STACK  MEMORY,  DESTROYS
                        ; REGISTERS  A,  E,  T,  &  P2,  AND  CALLS  THE  SUBROUTINES
                        ; 'CONBYT', 'CONNIB', & NIBL2 CALL 14 (PRTLN).

                        ; THE HEX NUMBER TO BE CONVERTED IS ASSUMED TO BE  STORED  IN
                        ; THE  NIBL2  VARIABLE H, AND THE ASCII EQUIVALENT OF THE HEX
                        ; NUMBER IS STORED IN THE MEMORY LOCATION POINTED TO  BY  THE
                        ; NIBL2  VARIABLE  D.  THE OUTPUT STRING CONSISTS OF 3 ASCII
                        ; CHARACTERS FOR A 2 DIGIT HEX NUMBER, AND 5 ASCII CHARACTERS
                        ; FOR A 4 DIGIT HEX NUMBER.  THE MSD IS STORED AT THE  LOWEST
                        ; MEMORY  LOCATION,  WHICH  IS  THE  BEGINNING  OF THE OUTPUT
                        ; STRING.


                        HEX2ASCI:
        8E3F 260010 PRT2:       LD      P2,=01000       ; POINT P2 TO NIBL2 VARIABLES

        8E42 8206               LD      EA,D,P2         ; LOAD CONTENTS OF NIBL2 VAR D
        8E44 B40300             ADD     EA,=3           ; ADD DISPL TO END OF STRING
                                                        ; + 1

        8E47 46                 LD      P2,EA           ; END OF STRING LOC + 1-->P2

        8E48 C401               LD      A,=1            ; STORE # OF BYTES TO CONVERT
        8E4A CB09               ST      A,BYTCTR,P3

        8E4C 740D               BRA     INIT            ; CONTINUE

        8E4E 260010 PRT4:       LD      P2,=01000       ; POINT P2 TO NIBL2 VARIABLES

        8E51 8206               LD      EA,D,P2         ; LOAD CONTENTS OF NIBL2 VAR D
        8E53 B40500             ADD     EA,=5           ; ADD DISPL TO END OF STRING
                                                        ; + 1

        8E56 46                 LD      P2,EA           ; END OF STRING LOC + 1-->P2

        8E57 C402               LD      A,=2            ; STORE # OF BYTES TO CONVERT
        8E59 CB09               ST      A,BYTCTR,P3

        8E5B C4A0   INIT:       LD      A,=0A0          ; STORE ASCII "SP" WHICH WILL
        8E5D CEFF               ST      A,@-1,P2        ; APPEAR AT END OF PRINTED
                                                        ; STRING, WITH B7=1 TO DENOTE
                                                        ; END OF STRING. DECREMENT P2

        8E5F 220010             PLI     P2,=01000       ; POINT P2 TO NIBL2 VAR &
                                                        ; SAVE OLD P2
```

```
     8E62 820E              LD      EA,H,P2       ; LOAD HEX # TO BE CONVERTED
     8E64 09                LD      T,EA          ; TO ASCII & SAVE IT IN T

     8E65 5E                POP     P2            ; RESTORE OLD P2

     8E66 20728E CONV1:     JSR     CONBYT        ; CONV HEX BYTE IN A REG TO
                                                  ; 2 ASCII CHARACTERS, STORE
                                                  ; THEM IN THE OUTPUT STRING
                                                  ; & DECREMENT P2 BY 2

     8E69 9B09              DLD     A,BYTCTR,P3   ; DECREMENT BYTE COUNT
     8E6B 6C04              BZ      PRINT         ; & EXIT IF COUNT = 0

     8E6D 0B        CONV2:  LD      EA,T          ; TRANSFER 16 BIT HEX
     8E6F 40                LD      A,E           ; # TO BE CONVERTED TO
                                                  ; TO EA REG, & PLACE
                                                  ; MSB IN THE A REG

     8E6F 74F5              BRA     CONV1         ; CONTINUE

     8E71 1E        PRINT:  CALL    14            ; PRINT OUT ASCII EQUIV
                                                  ; OF HEX # WITH TRAILING
                                                  ; SPACE AND NO CR/LF

     8E72 5C                RET                   ; RETURN
```

```
                                  .PAGE    'HEX2ASCI - CONBYT'

                            ; 'CONBYT' CONVERTS THE HEX BYTE PRESENT IN THE A REGISTER TO
                            ; ITS ASCII EQUIVALENT, AND STORES THE TWO  ASCII  CHARACTERS
                            ; CREATED  IN THE MEMORY LOCATIONS POINTED TO BY P2   P2 MUST
                            ; BE SET TO AVAILABLE RAM BEFORE THE SUBROUTINE  IS  ENTERED,
                            ; AND  P2  IS  DECREMENTED  BY 1 WHEN EACH ASCII CHARACTER IS
                            ; STORED.

                            ; THE SUBROUTINE DESTROYS REGISTERS A AND E,  AND  IS  EXITED
                            ; WITH  P2 POINTING TO THE MS ASCII DIGIT, WHICH IS STORED AT
                            ; THE LOWER MEMORY LOCATION.


   8E73 48       CONBYT: LD       E,A              ; SAVE BYTE TO BE
                                                   ; CONVERTED IN E REG

   8E74 207F8E           JSR      CONNIB  ,        ; CONVERT & STORE ASCII
                                                   ; EQUIV OF LS NIBBLE

   8E77 40               LD       A,E              ; RESTORE HEX BYTE TO A REG

   8E78 3C               SR       A                ; SHIFT MS NIBBLE TO LS
   8E79 3C               SR       A                ; NIBBLE POSITION
   8E7A 3C               SR       A
   8E7B 3C               SR       A

   8E7C 207F8E           JSR      CONNIB           ; CONVERT & STORE ASCII
                                                   ; EQUIV OF MS NIBBLE

   8E7F 5C               RET                       ; RETURN


   8E80 D40F     CONNIB: AND      A,=0F            ; MASK OFF LS 4 BITS

   8E82 FC0A             SUB      A,=0A            ; SUBTRACT X'A=10

   8E84 6404             BP       GE10             ; TEST RESULT

                                                   ; NIBBLE WAS 0 THRU 9
   8E86 F43A     LT10:   ADD      A,=X'3A          ; CONV NIBBLE TO ASCII

   8E88 7402             BRA      STORE            ; CONTINUE

                                                   ; NIBBLE WAS 10 THRU 15
   8E8A F441     GE10:   ADD      A,=X'41          ; CONV NIBBLE TO ASCII

   8E8C CEFF     STORE:  ST       A,@-1,P2         ; STORE ASCII EQUIV OF
                                                   ; NIBBLE & DECR P2 BY 1

   8E8E 5C               RET                       ; RETURN
 20                      .INCLD WRTAPE
```

```
                        .PAGE    'WRTAPE'

                  ; 'WRTAPE' INTERFACES THE INS8073 TO A CASSETTE RECORDER  FOR
                  ; STORAGE/RETRIEVAL  OF USER PROGRAMS.  PROGRAMS WHICH MAY BE
                  ; SAVED INCLUDE NIBL2 PROGRAMS AND ASSEMBLY LANGUAGE LM'S.

                  ; WHEN THE 'WRTAPE' SUBROUTINE IS USED  IN  CONJUNCTION  WITH
                  ; THE NIBL2 PROGRAM 'UTILITY', THE USER CAN SPECIFY THE BLOCK
                  ; OF  RAM  TO  BE WRITTEN ON THE TAPE.  THE TAPE FORMAT IS AS
                  ; FOLLOWS:
                  ;
                  ;   1) APPROXIMATELY 5 SECONDS OF 0'S WHICH SERVE AS LEADER
                  ;      SO THAT THE TAPE SPEED HAS TIME TO STABILIZE ON
                  ;      PLAYBACK.  THE LEADER ALSO ALLOWS THE RECEIVING
                  ;      PROGRAM TO PROPERLY SYNC TO THE CLOCK PULSES.
                  ;   2) ID CHARACTER=X'A5 WHICH IDENTIFIES THE START OF
                  ;      EACH RECORD.
                  ;   3) A BYTE WHICH SPECIFIES THE RECORD TYPE:
                  ;      DATA RECORD=X'01  END RECORD=X'03
                  ;   4) A BYTE WHICH IDENTIFIES THE TOTAL NUMBER
                  ;      OF DATA BYTES IN EACH RECORD, N.
                  ;      N CAN RANGE FROM 1 TO 256. (0 - 255)
                  ;   5) THE LSB OF THE STARTING ADDRESS WHERE THE DATA
                  ;      RECORD IS TO BE STORED.
                  ;   6) THE MSB OF THE STARTING ADDRESS WHERE THE DATA
                  ;      RECORD IS TO BE STORED.
                  ;   7) 1 - 256 PROGRAM BYTES
                  ;   8) A SINGLE BYTE CHECKSUM (IN 2'S COMPLEMENT FORM)
                  ;      OF ALL BYTES CONTAINED IN THE RECORD EXCEPT FOR
                  ;      THE ID CHARACTER


                  ; DISPLACEMENTS RELATIVE TO P3:

0000                    .SET    NB,0           ; NB=REMAINING # OF PGM BYTES
                                               ; TO BE WRITTEN (2 BYTES)
0002                    .SET    N,2            ; N=# DATA BYTES IN DATA REC
0003                    .SET    CKSUM,3        ; CKSUM IS THE RECORD CHECKSUM
                                               ; ACCUMULATED IN MEMORY
0004                    .SET    WRCTR,4        ; BIT COUNTER FOR
                                               ; THE 'WRCHAR' SUBROUTINE
0005                    .SET    LDRCTR,5       ; LEADER COUNTER FOR THE
                                               ; SNDLDR ROUTINE (2 BYTES)


                  ; DISPLACEMENTS RELATIVE TO P2:

002E                    .SET    X,46           ; LOC OF NIBL2 VARIABLE X
                                               ; (MEMORY STARTING ADDR)
0030                    .SET    Y,48           ; LOC OF NIBL2 VARIABLE Y
                                               ; (MEMORY ENDING ADDR)
```

```
        FF00              .SET     SRAM,0FF00     ; SCRATCH RAM TO WHICH P3
                                                  ; POINTS FOR THE 'WRTAPE'
                                                  ; SUBROUTINE


   8E8F 39FE     WRTAPE:  AND      S,=0FE         ; DISABLE INTERRUPTS

   8E91 2700FF            LD       P3,=SRAM       ; POINT P3 TO SCRATCH RAM

   8E94 39FB              AND      S,=0FB         ; SET F2=0

   8E96 3B08              OR       S,=08          ; SET F3=1

                 ; CALCULATE AND STORE NB = # OF DATA BYTES TO BE WRITTEN

   8E98 8230              LD       EA,Y,P2        ; LOAD ENDING ADDRESS
                                                  ; (NIBL2 Y) INTO EA REG
   8E9A BA2E              SUB      EA,X,P2        ; SUBTRACT STARTING
                                                  ; ADDRESS (NIBL2 X)
   8E9C B40100            ADD      EA,=01         ; ADD 1
   8E9F 8B00              ST       EA,NB,P3       ; SAVE NB


                 ; SET PTR P2 = STARTING ADDRESS WHERE DATA IS TO BE STORED

   8EA1 822E     LDP2:    LD       EA,X,P2        ; LOAD STARTING ADDRESS
                                                  ; INTO EA REG
   8EA3 46                LD       P2,EA          ; SET P2=STARTING ADDRESS



                 ; SEND LEADER ROUTINE

                 ; THIS ROUTINE  TRANSMITS  APPROXIMATELY  5  SECONDS  OF  0'
                 ; (APPROX  2500  @ 500 BAUD) TO ACT AS LEADER, ALLOW THE TAP
                 ; TO SETTLE ON PLAYBACK, AND TO  ALLOW  PROPER  SYNC  TO  TH
                 ; CLOCK PULSES.

                 ; SNDLDR ROUTINE PARAMETERS:

        09C4              .SET     LDRCNT,2500    ; # OF CLK PULSES IN LEADER
        005E              .SET     BITDLY,94      ; DELAY COUNT TO PRODUCE 1
                                                  ; BIT DELAY

   8EA4 84C409 SNDLDR: LD          EA,=LDRCNT     ; LOAD LEADER COUNT
   8EA7 8B05              ST       EA,LDRCTR,P3   ; AND STORE IT

   8EA9 20538F LOOPA:     JSR      PULSE          ; WRITE CLK PULSE

   8EAC C45E              LD       A,=BITDLY      ; LOAD DELAY COUNT
   8EAE 20F78F            JSR      DELAY          ; DELAY 1 BIT TIME

   8EB1 8305              LD       EA,LDRCTR,P3   ; LOAD LEADER COUNT
   8EB3 BC0100            SUB      EA,=1          ; DECR LEADER COUNT
```

```
8EB6 8B05              ST      EA,LDRCTR,P3 ; STORE NEW LEADER COUNT

8EB8 58                OR      A,E          ; TEST FOR LEADER COUNT
8EB9 7CEF              BNZ     LOOPA        ; =0.



8EBB 84FF00 RECORD:    LD      EA,=255      ; LOAD 255
8EBF BB00              SUB     EA,NB,P3     ; SUBTRACT NB
8EC0 3F                RRL     A            ; GET CY INTO A7
8EC1 640B              BP      GT255        ; AND TEST IT

8EC3 8300   LE255:     LD      EA,NB,P3     ; CY=1. NB<=255
8EC5 CB02              ST      A,N,P3       ; SET N=NB
8EC7 840000            LD      EA,=0        ; SET NB=0
8ECA 8B00              ST      EA,NB,P3
8ECC 740B              BRA     NXT1         ; CONTINUE

8ECE C400   GT255:     LD      A,=0         ; CY=0. NB>255
8ED0 CB02              ST      A,N,P3       ; SET N=0

8ED2 8300              LD      EA,NB,P3     ; SET NB=NB-256
8ED4 BC0001            SUB     EA,=256
8ED7 8B00              ST      EA,NB,P3

8ED9 C302   NXT1:      LD      A,N,P3       ; LOAD N INTO A REG
8EDB F401              ADD     A,=01        ; ADD REC TYPE=X'01
8EDD CB03              ST      A,CKSUM,P3   ; STORE INTO CHKSUM

8EDF C4A5              LD      A,=X'A5      ; LOAD ID CHAR=X'A5
8EE1 20298F            JSR     WRCHAR       ; WRITE CHAR ON TAPE

8EE4 C401              LD      A,=01        ; LOAD DATA REC TYPE
8EE6 20298F            JSR     WRCHAR       ; =X'01 & WRITE ON TAPE

8EE9 C302              LD      A,N,P3       ; LOAD # OF BYTES IN DATA REC
8EEB 20298F            JSR     WRCHAR       ; & WRITE ON TAPE

8EEE 32                LD      EA,P2        ; LD STARTING ADDR INTO EA
8EEF 20298F            JSR     WRCHAR       ; WRITE LSB ON TAPE

8EF2 F303              ADD     A,CKSUM,P3   ; ADD CKSUM TO LSB OF STARTING
8EF4 CB03              ST      A,CKSUM,P3   ; ADDRESS & STORE NEW CKSUM

8EF6 32                LD      EA,P2        ; LD STARTING ADDR INTO EA
8EF7 40                LD      A,E          ; MSB OF STARTING ADDR-->A REG
8EF8 20298F            JSR     WRCHAR       ; WRITE MSB ON TAPE

8EFB F303              ADD     A,CKSUM,P3   ; ADD CKSUM TO MSB OF STARTING
8EFD CB03              ST      A,CKSUM,P3   ; ADDRESS & STORE NEW CKSUM

8EFF C601   LOOPD:     LD      A,@+1,P2     ; LD DATA BYTE & INCR PTR
8F01 20298F            JSR     WRCHAR       ; WRITE DATA BYTE ON TAPE
8F04 F303              ADD     A,CKSUM,P3   ; ADD CHAR TO CKSUM
```

```
        8F06 CB03            ST      A,CKSUM,P3    ; STORE NEW CKSUM

        8F08 9B02            DLD     A,N,P3        ; SET N=N-1
        8F0A 7CF3            BNZ     LOOPD         ; & LOOP IF N<>0

        8F0C C303            LD      A,CKSUM,P3    ; LOAD CKSUM INTO A
        8F0E E4FF            XOR     A,=0FF        ; TAKE 2'S COMPLEMENT
        8F10 F401            ADD     A,=01         ; OF CKSUM AND
        8F12 20298F          JSR     WRCHAR        ; WRITE IT ON THE TAPE

        8F15 8300            LD      EA,NB,P3      ; LOAD NB & TEST FOR 0
        8F17 58              OR      A,E           ; WRITE NEXT RECORD IF
        8F18 7CA1            BNZ     RECORD        ; NB<>0

                      ; WRITE END RECORD ON THE TAPE

        8F1A C4A5            LD      A,=X'A5       ; NB=0
        8F1C 20298F          JSR     WRCHAR        ; WRITE ID CHAR ON TAPE

        8F1F C403            LD      A,=03         ; WRITE END RECORD TYPE
        8F21 20298F          JSR     WRCHAR        ; =X'03 ON TAPE

        8F24 C4FD            LD      A,=X'FD       ; WRITE 2'S COMPLEMENT OF
        8F26 20298F          JSR     WRCHAR        ; CKSUM ON TAPE
        8F29 5C              RET                   ; RETURN
```

```
                                   .PAGE    'WRTAPE - WRCHAR'

                          ; 'WRCHAR' WRITES THE 8 BIT CHAR PRESENT IN THE A REG ON  THE
                          ; TAPE.   THE  PROGRAM  DESTROYS  REGISTER  E,  ASSUMES  THAT
                          ; POINTER P3 IS POINTING TO 1 BYTE OF AVAILABLE RAM  (WRCTR),
                          ; AND CALLS THE 'DELAY' SUBROUTINE.

                          ; SUBROUTINE PARAMETERS INCLUDE:

                                  ;          WRCTR             LOC WHERE BIT COUNT WILL
                                  ;                            BE STORED RELATIVE TO P3
         001A                     .SET       HLFDLY,26       ; COUNT FOR 1/2 BIT DELAY
         001A                     .SET       ENDDLY,26       ; COUNT FOR END OF BIT DELAY
         005F                     .SET       FULDLY,95       ; COUNT FOR 1 BIT DELAY



    8F2A 48       WRCHAR: LD         E,A             ; SAVE CHAR IN E REG

    8F2B C408             LD         A,=08           ; SET BIT COUNT=8
    8F2D CB04             ST         A,WRCTR,P3

    8F2F 40       SHIFT:  LD         A,E             ; XFER CHAR TO A REG
    8F30 3E               RR         A               ; ROTATE LSB TO BIT 7
                                                     ; WHERE IT CAN BE SENSED
    8F31 48               LD         E,A             ; SAVE ROTATED CHAR IN E

    8F32 6412             BP         SENDO           ; TEST BIT TO BE WRITTEN

    8F34 20538F SEND1:    JSR        PULSE           ; BIT=1. SEND CLOCK PULSE

    8F37 C41A             LD         A,=HLFDLY       ; SET DLY COUNT=1/2 BIT TIME
    8F39 20F78F           JSR        DELAY           ; DELAY TO MIDDLE OF BIT

    8F3C 20538F           JSR        PULSE           ; WRITE DATA BIT=1

    8F3F C41A             LD         A,=ENDDLY       ; DELAY TO END OF
    8F41 20F78F           JSR        DELAY           ; BIT TIME

    8F44 7408             BRA        DECCNT          ; CONTINUE

    8F46 20538F SENDO:    JSR        PULSE           ; BIT=0. SEND CLOCK PULSE

    8F49 C45F             LD         A,=FULDLY       ; SET DLY COUNT=1 BIT TIME
    8F4B 20F78F           JSR        DELAY           ; DELAY 1 BIT TIME

    8F4E 9B04     DECCNT: DLD        A,WRCTR,P3      ; DECREMENT BIT COUNT

    8F50 7CDD             BNZ        SHIFT           ; REPEAT UNTIL BIT COUNT=0

    8F52 40               LD         A,E             ; RESTORE ORIG CHAR TO A REG

    8F53 5C               RET                        ; RETURN
```

```
                             .PAGE    'WRTAPE - PULSE'

                       ; 'PULSE' WRITES 1 CLOCK OR DATA PULSE ON THE TAPE, CALLS THE
                       ; 'DELAY' SUBROUTINE, AND ASSUMES THAT FLAGS  F2  &  F3  HAVE
                       ; BEEN INITIALIZED TO THE STATE F2=0 AND F3=1.


                       ; SUBROUTINE PARAMETERS INCLUDE:

        000F                 .SET     D1,15           ; DELAY WHICH SETS DURATION
                                                      ; OF POSITIVE EXCURSION
        000F                 .SET     D2,15           ; DELAY WHICH SETS DURATION
                                                      ; OF NEGATIVE EXCURSION

                       ; OUTPUT PULSE GENERATED BY COMBINING
                       ; F2 & F3 OUTPUTS APPEARS AS FOLLOWS:
                       ;
                       ;           ++++++
                       ;           + D1 +
                       ;      ++++++    +     +++++++
                       ;                + D2 +
                       ;                ++++++


    8F54 3B04   PULSE:   OR       S,=04           ; SET F2=1. (F3=1)

    8F56 C40F            LD       A,=D1           ; SET DELAY COUNT=D1
    8F58 20F78F          JSR      DELAY           ; DELAY FOR D1

    8F5B 39F3            AND      S,=0F3          ; SET F2=F3=0

    8F5D C40F            LD       A,=D2           ; SET DELAY COUNT=D2
    8F5F 20F78F          JSR      DELAY           ; DELAY FOR D2

    8F62 3B08            OR       S,=08           ; SET F3=1. (F2=0)

    8F64 5C              RET                      ; RETURN
  21                     .INCLD RDTAPE
```

```
                          .PAGE    'RDTAPE'

                    ; 'RDTAPE' INTERFACES THE INS8073 TO A CASSETTE RECORDER  FOR
                    ; STORAGE/RETRIEVAL  OF USER PROGRAMS.  PROGRAMS WHICH MAY BE
                    ; SAVED AND RETRIEVED INCLUDE  NIBL2  PROGRAMS  AND  ASSEMBLY
                    ; LANGUAGE LM'S.

                    ; THE 'RDTAPE' SUBROUTINE REQUIRES 6 BYTES OF SCRATCHPAD  RAM
                    ; AND CALLS THE SUBROUTINES 'GETBIT' AND 'RCVCHR'.

                    ; WHEN A TAPE RESIDENT PROGRAM IS READ INTO RAM, THE USER MAY
                    ; SPECIFY AN OPTIONAL DISPLACEMENT  WHICH  IS  ADDED  TO  THE
                    ; STARTING  ADDRESS OF EACH DATA RECORD.  THIS FEATURE ALLOWS
                    ; ASSEMBLY LANGUAGE LM'S AND NIBL2 PROGRAMS TO BE LOADED INTO
                    ; MEMORY AT LOCATIONS SPECIFIED AT LOAD TIME.

                    ; THE FORMAT OF THE DATA WRITTEN ON THE TAPE IS AS FOLLOWS:
                    ;
                    ;    1) APPROXIMATELY 5 SECONDS OF 0'S WHICH SERVE AS LEADER
                    ;       SO THAT THE TAPE SPEED HAS TIME TO STABILIZE ON
                    ;       PLAYBACK.  THE LEADER ALSO ALLOWS THE RECEIVING
                    ;       PROGRAM TO PROPERLY SYNC TO THE CLOCK PULSES.
                    ;    2) ID CHARACTER=X'A5 WHICH IDENTIFIES THE START
                    ;       OF EACH RECORD.
                    ;    3) A BYTE WHICH SPECIFIES THE RECORD TYPE:
                    ;       DATA RECORD=X'01  END RECORD=X'03
                    ;    4) A BYTE WHICH IDENTIFIES THE TOTAL NUMBER
                    ;       OF DATA BYTES IN EACH RECORD, N.
                    ;       N CAN RANGE FROM 1 TO 256. (0 - 255)
                    ;    5) THE LSB OF THE STARTING ADDRESS WHERE THE DATA
                    ;       RECORD IS TO BE STORED.
                    ;    6) THE MSB OF THE STARTING ADDRESS WHERE THE DATA
                    ;       RECORD IS TO BE STORED.
                    ;    7) 1 - 256 PROGRAM BYTES
                    ;    8) A SINGLE BYTE CHECKSUM (IN 2'S COMPLEMENT FORM)
                    ;       OF ALL BYTES CONTAINED IN THE RECORD
                    ;       EXCEPT FOR THE ID CHARACTER


                    ; DISPLACEMENTS RELATIVE TO PTR P3:

        0000              .SET     CKSUM,0        ; CHECKSUM FORMED IN MEMORY
        0001              RCVCTR   = 1            ; # OF CHAR BITS RECEIVED
                                                  ; IN 'RCVCHR' SUBROUTINE
        0002              STADR    = 2            ; RECORD STARTING ADDRESS
                                                  ; (2 BYTES)
        0004              .SET     N,4            ; # OF DATA BYTES IN DATA REC
        0005              SCOUNT   = 5            ; SAMPLE COUNT (# OF SAMPLES
                                                  ; IN 'GETBIT' SUBROUTINE)

                    ; DISPLACEMENTS RELATIVE TO PTR P2:

        002F              .SET     X,46           ; STARTING ADDR DISPLACEMENT
```

```
                                                  ; (NIBL2 VARIABLE Z)

       0032              .SET    Z,50             ; CHECKSUM ERROR FLAG
                                                  ; (NIBL2 VARIABLE Z)


       FF00              .SET    SRAM,0FF00       ; SCRATCH RAM TO WHICH P3
                                                  ; POINTS


  8F65 39FE    RDTAPE:   AND     S,=0FE           ; DISABLE INTERRUPTS
  8F67 2700FF            LD      P3,=SRAM         ; POINT P3 TO SCRATCH RAM

  8F6A C400    REPEAT:   LD      A,=0             ; LOAD 0
  8F6C 48                LD      E,A              ; SET CHAR=0 IN E REG
  8F6D CB00              ST      A,CKSUM,P3       ; SET CKSUM=0

  8F6F 20018F  SYNCLP:   JSR     GETBIT           ; SHIFT BIT INTO CHAR
                                                  ; WHICH IS RETURNED IN
                                                  ; A REG AND E REG
  8F72 E4A5              XOR     A,=X'A5          ; TEST FOR CHAR=X'A5
  8F74 7CF9              BNZ     SYNCLP           ; GOTO SYNCLP IF CHAR<>X'A5

                                                  ; CHAR=X'A5
  8F76 20E78F            JSR     RCVCHR           ; GET RECORD TYPE IN
                                                  ; A REG & E REG
  8F79 F300              ADD     A,CKSUM,P3       ; ADD CKSUM TO RECORD TYPE
  8F7B CB00              ST      A,CKSUM,P3       ; STORE NEW CKSUM

  8F7D 40                LD      A,E              ; LOAD RECORD TYPE INTO A REG

  8F7E E401              XOR     A,=01            ; TEST A REG FOR DATA RECORD
  8F80 7C38              BNZ     EREC             ; DATA RECORD XMITTED IF A=01

                                                  ; A=01. DATA RECORD BEING RECD
  8F82 20E78F  DREC:     JSR     RCVCHR           ; GET N=# OF DATA BYTES IN
                                                  ; THE RECORD IN THE A REG

  8F85 CB04              ST      A,N,P3           ; SAVE N

  8F87 F300              ADD     A,CKSUM,P3       ; ADD CKSUM TO N
  8F89 CB00              ST      A,CKSUM,P3       ; STORE NEW CKSUM

  8F8B 20E78F            JSR     RCVCHR           ; ADDRL-->A REG
  8F8E 0A                PUSH    A                ; SAVE ADDRL ON STACK

  8F8F F300              ADD     A,CKSUM,P3       ; ADD CKSUM TO ADDRL
  8F91 CB00              ST      A,CKSUM,P3       ; STORE NEW CKSUM

  8F93 20E78F            JSR     RCVCHR           ; ADDRH-->A REG & E REG
  8F96 F300              ADD     A,CKSUM,P3       ; ADD CKSUM TO ADDRH
  8F98 CB00              ST      A,CKSUM,P3       ; STORE NEW CKSUM

  8F9A 38                POP     A                ; ADDRL-->A REG. (ADDRH
                                                ; IS ALREADY IN E REG)
    8F9B B22E            ADD     EA,X,P2        ; ADD IN DISPLACEMENT
                                                ; (NIBL2 VARIABLE X)

    8F9D 56              PUSH    P2             ; SAVE NIBL2 VARIABLE PTR
    8F9E 46              LD      P2,EA          ; XFER REC STARTING ADDR TO P2

    8F9F 20E78F GETDTA:  JSR     RCVCHR         ; DATA BYTE-->A REG
    8FA2 CE01            ST      A,@+1,P2       ; STORE BYTE & INCR PTR
    8FA4 F300            ADD     A,CKSUM,P3     ; ADD CKSUM TO DATA BYTE
    8FA6 CB00            ST      A,CKSUM,P3     ; STORE NEW CKSUM
    8FA8 9B04            DLD     A,N,P3         ; DECR CHAR COUNT,N, &
                                                ; LOAD INTO A REG
    8FAA 70F3            BNZ     GETDTA         ; GET NEXT DATA BYTE IF
                                                ; COUNT,N,IS NOT 0

                                                ; CHAR COUNT=N=0
    8FAC 5E              POP     P2             ; RESTORE NIBL2 VARIABLE PTR
    8FAD 20E78F          JSR     RCVCHR         ; TAPE CKSUM-->REG A
    8FB0 F300            ADD     A,CKSUM,P3     ; ADD CKSUM STORED IN
                                                ; MEMORY TO TAPE CKSUM
    8FB2 6CB6            BZ      REPEAT         ; IF A REG=0 BOTH CHECKSUMS
                                                ; MATCH; GET A NEW RECORD

                                                ; A REG <> 0. CHECKSUMS DIFFER
    8FB4 840100 SERR:    LD      EA,=01         ; SET ERROR FLAG,
    8FB7 8A32            ST      EA,Z,P2        ; NIBL2 VARIABLE Z, = 1
    8FB9 5C              RET                    ; RETURN

                                                ; A REG <> 0. END RECORD RECD
    8FBA 20E78F EREC:    JSR     RCVCHR         ; TAPE CKSUM-->A REG
    8FBD F300            ADD     A,CKSUM,P3     ; ADD CKSUM STORED IN
                                                ; MEMORY TO TAPE CKSUM
    8FBF 70F3            BNZ     SERR           ; IF A REG=0 CHECKSUMS MATCH

                                                ; A REG=0. CHECKSUMS MATCH
    8FC1 5C              RET                    ; RETURN
```

```
                          .PAGE    'RDTAPE - GETBIT'

                    ; 'GETBIT' RECEIVES 1 BIT INTO BIT 7 OF THE E REGISTER.  (THE
                    ; E REGISTER MUST BE SET TO  0  BEFORE  A  CHARACTER  CAN  BE
                    ; FORMED).   'GETBIT'  IS  CALLED  8  TIMES  BY  THE  'RDCHR'
                    ; SUBROUTINE IN ORDER TO RECIEVE AN 8 BIT CHARACTER INTO  THE
                    ; E  REGISTER.   'GETBIT'  IS  ALSO  REPEATEDLY CALLED BY THE
                    ; 'RDTAPE' PROGRAM SYNCHRONIZATION LOOP IN  ORDER  TO  LOCATE
                    ; THE START OF RECORD CHARACTER (X'A5').

                    ; INVERTED DATA AND CLOCK PULSES ARE RECEIVED ON THE  SB INPUT
                    ; (WHEN SB=0 DATA/CLOCK ARE PRESENT).

                    ; 'GETBIT' ASSUMES THAT P3 IS POINTING TO 1 BYTE OF AVAILABLE
                    ; SCRATCHPAD RAM (SCOUNT), AND CALLS THE SUBROUTINE  'DELAY'.

                    ; GETBIT SUBROUTINE PARAMETERS:

      0039                HDLY1 = 57             ; DELAY TO START OF FIRST  SAMPLE

                          ; SCOUNT IS THE SAMPLE COUNT (# OF SAMPLES
                          ; TAKEN BEFORE A '0' DATA BIT IS RETURNED)


  8FC2 C40C     GETBIT: LD      A,=12          ; SET SAMPLE COUNT=?
  8FC4 CB05             ST      A,SCOUNT,P3

  8FC6 06       GETCLK: LD      A,S            ; WAIT FOR CLOCK PULSE ON
                                               ; SB INPUT

                  ;;;;  OR      S,=04          ; ***PULSE F2********
                  ;;;;  AND     S,=0FB         ; *******************

  8FC7 D420             AND     A,=020         ; MASK OFF SB
  8FC9 7CFB             BNZ     GETCLK         ; WAIT UNTIL SB GOES LOW

  8FCB C439             LD      A,=HDLY1       ; CLOCK PULSE IS PRESENT
  8FCD 20F78F           JSR     DELAY          ; DELAY TO START OF
                                               ; SAMPLE TIME
  8FD0 06       SMPL:   LD      A,S            ; SAMPLE INVERTED BIT ON SB

                  ;;;;  OR      S,=08          ; ***PULSE F3*******
                  ;;;;  AND     S,=0F7         ; ******************

  8FD1 D420             AND     A,=020

  8FD3 6C08             BZ      RET1           ; TEST SAMPLED BIT =0 OR 1

                                               ; SAMPLED BIT=0
  8FD5 9B05     SMPLO:  DLD     A,SCOUNT,P3    ; DECREMENT SAMPLE COUNT &
  8FD7 7CF7             BNZ     SMPL           ; CONTINUE IF COUNT<>0

                                               ; FINAL VALUE OF DATA BIT=0
  8FD9 40       RETO:   LD      A,E            ; LOAD CHARACTER
```

```
     8FDA 3C                SR      A           ; INTO THE A REG & SHIFT
                                                ; IT RIGHT BY 1 BIT.
                                                ; BIT 7 =0 BY DEFAULT

     8FDB 48                LD      E,A         ; PUT CHAR INTO E REG
     8FDC 5C                RET                 ; RETURN

                                                ; FINAL VALUE OF DATA BIT=1
     8FDD 06        RET1:   LD      A,S         ; WAIT UNTIL SB=1
     8FDE D420              AND     A,=020      ; (DATA PULSE GOES AWAY)
     8FE0 6CFB              BZ      RET1

     8FE2 40                LD      A,E         ; LOAD CHARACTER
     8FE3 3C                SR      A           ; INTO THE A REG & SHIFT
     8FE4 DC80              OR      A,=080      ; IT RIGHT BY 1 BIT
                                                ; SET BIT 7 =1

     8FE6 48                LD      E,A         ; PUT CHAR INTO E REG
     8FE7 5C                RET                 ; RETURN
```

```
                           .PAGE   'RDTAPE - RDVCHR'

                    ; 'RDVCHR' RECEIVES ONE 8 BIT CHARACTER INTO THE  A  REGISTER
                    ; AND THE E REGISTER.

                    ; 'RCVCHR' ASSUMES THAT PTR P3  IS  POINTING  TO  1  BYTE  OF
                    ; AVAILABLE SCRATCHPAD RAM (RCVCTR), AND CALLS THE SUBROUTINE
                    ; 'GETBIT'  8  TIMES  IN  ORDER  TO  RECEIVE A COMPLETE 8 BIT
                    ; CHARACTER.

                    ; PARAMETERS FOR 'RCVCHR' INCLUDE:

                                    ; RCVCTR IS THE COUNTER WHICH COUNTS
                                    ; THE NUMBER OF BITS RECEIVED

     8FE8 C408   RCVCHR: LD      A,=08         ; SET BIT COUNT =8
     8FEA CB01           ST      A,RCVCTR,P3

     8FEC C400           LD      A,=0          ; CLEAR THE E REG WHERE
     8FEE 48             LD      E,A           ; CHAR WILL BE FORMED

     8FEF 20C18F LOOP2:  JSR     GETBIT        ; GET 1 BIT INTO E REG

     8FF2 9B01           DLD     A,RCVCTR,P3   ; DECREMENT BIT COUNT

     8FF4 70F9           BNZ     LOOP2         ; CONTINUE UNTIL COUNT=0

     8FF6 40             LD      A,E           ; PUT CHAR INTO A REG

     8FF7 5C             RET                   ; RETURN
  22                     .INCLD DELAY
```

```
                            .PAGE    'DELAY'

                    ; 'DELAY' GENERATES A DELAY BY  DECREMENTING  A  DELAY  COUNT
                    ; WHICH HAS BEEN PREVIOUSLY LOADED INTO THE A REGISTER.

                    ; WHEN EXECUTED FROM EXTERNAL MEMORY, THE  TOTAL  TIME  DELAY
                    ; (UCYCLES)   GENERATED  BY  THE  SUBROUTINE,  INCLUDING  THE
                    ; PREVIOUSLY  EXECUTED  INSTRUCTIONS  'JSR   DELAY   &   'LD
                    ; A,=DELAYCOUNT', IS AS FOLLOWS:
                    ;
                    ;        TOTAL DELAY = 39 + 14 * DELAYCOUNT
                    ;
                    ; WHERE DELAYCOUNT RANGES FROM 1 TO 255.

      8FF8 FC01    DELAY:   SUB      A,=01          ; DECREMENT PREVIOUSLY
                                                     ; LOADED DELAY COUNT
      8FFA 70FC             BNZ      DELAY          ; LOOP UNTIL COUNT = 0

      8FFC 5C               RET                      ; RETURN

   23      0000             .END
```

```
8070 ASSEMBLER   REV-A   06/06/79
ASS070  8070 UTILITY SUBROUTINES



ADRMSG  8E11           ASCILD  8D81  *        ATOF1   8E24           ATOF2   8F39
B110    8D81  *        B4800   8D88  *        BITDLY  005E           BYTCTR  0007
CKSUM   0000           CONBYT  8E73           CONNIB  8E80           CONV1   8F66
CONV2   8E6D  *        CTR     0004           D       0006           D1      000F
D2      000F           DECCNT  8F4E           DELAY   8FF8           DREC    8FA2  *
DTAREC  8DDA           ENDDLY  001A           EREC    8FBA           FDELAY  FFEC
FULDLY  005F           GE10    8E8A           GECO    0933           GETBIT  8FC
GETBYT  8E17           GETCLK  8FC6           GETDTA  8F9F           GT255   8ECF
H       000F           HDLY1   0039           HEX2AS  8E3F  *        HLFDLY  0013
INIT    8E5B           L1      8E06           LDP2    8EA1  *        LDRCNT  09C4
LDRCTR  0005           LE255   8EC3  *        LOOP1   8D92           LOOP2   8FFF
LOOPA   8EA9           LOOPD   8EFF           LT10    8E86  *        MCKSUM  0001
MSN     0003           N       0004           NB      0000           NBYTES  0000
NEXT    8D8D           NOTDTA  8DEA           NSTOR1  8D95           NSTOR2  8E1
NSTOR3  8F2F           NXT1    8FD9           NXTBYT  8DDB           ORNTPL  [illegible]
PRINT   8E71           PRT2    8E3F  *        PRT4    8E4E           PULSE   8F54
RCVCHR  8FE8           RCVCTR  0001           RDTAPE  8F65  *        RECORD  8FBB
RECTST  8DFC           RECTYP  0002           REPEAT  8F6A           RET0    8FD2  *
RET1    8FDD           SCOUNT  0005           SEND0   8F46           SEND1   8F34  *
SERR    8FB4           SHIFT   8F2F           SHIFT1  8E26           SMPL    8FD0
SMPLO   8FD5  *        SNDLDR  8EA4  *        SRAM    FF00           STADR   0002  *
STORE   8E8C           SYNCLP  8F6F           WRCHAR  8F2A           WRCTR   0004
WRTAPE  8E8F  *        X       002E           Y       0030           Z       0032


 NO ERROR LINES
SOURCE CHECKSUM = FEF2
OBJECT CHECKSUM = 0DA5
INPUT FILE     1:ASSYLANG.SRC ON UT8070
LISTING FILE   1:ASSYLANG.LST ON UT8070
OBJECT FILE    1:ASSYLANG.LM ON UT8070
```