PROGRAMMING THE NICOLET 1080
STORED PROGRAM COMPUTER

A Course in Programming for the Beginner

Nicolet Instrument Corporation
5225 Verona Road
Madison, Wisconsin 53711

Revised December 1974

TABLE OF CONTENTS

I. BASIC NUMERICAL CONCEPTS

A. Introduction

The Nicolet Instrument Corporation 1080 computer is uniquely designed. It incorporates the best features of the hard-wired signal averager for data acquisition and the versatility of the general purpose computer for data manipulation. The 1080 actually consists of two separate processors utilizing the same memory and many of the same registers:

1. A wired program processor performs the data acquisition functions of analog-to-digital signal conversion, addition of numbers, storage of data points in memory, timing and counting of sweeps and display.

2. A powerful general purpose computer, which can be used to perform Fourier transforms, theoretical calculations, and data manipulations.

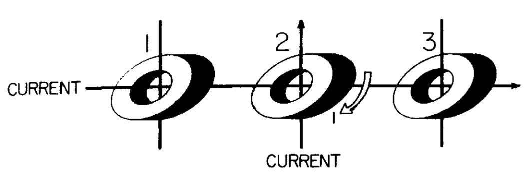
This manual will discuss the programming and use of the stored program processor section of the 1080. References will be made, however, to the relationship of stored and wired program throughout.

B. Architecture

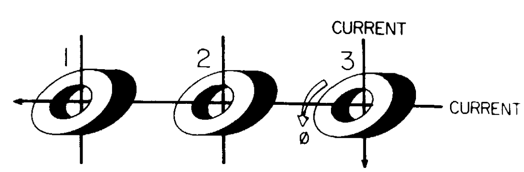
1. Description of Memory

The 1080 memory consists of small doughnut shaped pieces of ferrite called cores, each about the size of a period. Each core can be magnetized in one of two directions, representing 1 or Ø, yes or no, on or off, true or false, etc. These bistable devices are used to represent "binary digits" or simply, "bits."

The magnetic state of a particular core is changed by applying sufficient current through wires strung through each core. In order that specific cores may be changed independently, two sets of wires called x- and y-drivers are strung through each core. Half of the necessary current is then applied through each of the drivers, causing the change in magnetization to take place only in the core where the two wires intersect. The left hand drawing shows the conditions for "writing" a 1 into one core and the right hand one the conditions needed to write a Ø into one core.



Writing a 1 into core number 2



Writing a Ø into core number 3

The contents of a core can be examined by writing a zero into it and observing whether any change in magnetic flux occurs by this process. If a change occurs, the core was previously in the 1 state. If no change occurs, the core was already in the zero state. This change in flux is detected by a third wire strung through the cores, called a sense wire. A change in magnetic flux will induce a current in the sense wire which is then amplified, detected and used to set a one into a more permanent two-state device called a flip-flop. Since memory must be destroyed to examine it, the next step that the computer always performs is to copy the result back into memory from the flip-flop. The flip-flop is unchanged by this process.

2. Computer Words

Rather than just utilizing endless strings of binary bits, the digital computer decodes a certain size group of bits as one logical unit, or word. Typical word lengths of various minicomputers are 8, 12, 16 or 18 bits. Because the 1080 is a signal averager as well as a data processor, the word length adopted was 20 bits, since this provides an exceptionally large number of bits to accumulate signals containing coherent noise contributions.

Since each computer word contains twenty bits, it can be used to *represent* numbers ranging from Ø to $2^{2\emptyset}-1$. Numbers are represented in binary format, where each digit is either a one or a zero. The right-most bit represents 1 or $2^{\emptyset}$ and the left-most bit 2^{19}. If all bits are ones, the resulting number equals $20^{20}-1$, since the addition of one to this number would cause the zeroing of all bits in the word and the carry-out would set an imaginary 21st bit. The twenty bit computer word is illustrated below. In order to remind us of the relationship between each bit and a power of two, the bits are numbered from Ø through 19 from right to left.

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
524,288	262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1

C. Number Systems

1. Binary Notation

Nearly all digital computers in use today use the binary number (base 2) internally. Only the digits (or "bits") 0 and 1 are used in this number system. One can count from one to ten in binary as follows: 0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010. Given a binary number, it can be converted from binary (base 2) to decimal (base 10) by simply considering each digit to be a multiplier of a power of two. Thus the number 1010 can be considered as:

$$
\begin{array}{rcrcr}
1 \times 2^3 & = & 1 \times 8 & = & 8 \\
+0 \times 2^2 & = & 0 \times 4 & = & 0 \\
+1 \times 2^1 & = & 1 \times 2 & = & 2 \\
+0 \times 2^0 & = & 0 \times 1 & = & \underline{0} \\
 & & & & 10
\end{array}
$$

It is not necessary to convert all numbers from binary to decimal to perform operations with them, however. Let us examine the addition of numbers in binary notation.

$$
\begin{array}{r}
1 \\
\underline{+0} \\
1
\end{array}
\qquad
\begin{array}{r}
1 \\
\underline{+1} \\
10
\end{array}
\qquad
\begin{array}{r}
10 \\
\underline{+10} \\
100
\end{array}
\qquad
\begin{array}{r}
10111 \\
\underline{+10001} \\
101000
\end{array}
\qquad
\begin{array}{r}
01110111010110111011 \\
\underline{+01101101101010100110} \\
11100101000001100001
\end{array}
$$

As you can see from the above, the manipulation of even small binary numbers rapidly becomes quite cumbersome and the manipulation of 20-bit numbers boggles the mind. For this reason, it is customary to use a sort of shorthand method of representing binary numbers. This method contains as much information, is readily convertible to binary and is much easier to assimilate. This shorthand is called the octal or base-8 number system.

2. Octal Number System

The table shown below compares the octal, decimal and binary number systems:

Decimal	Octal	Binary
0	0	000
1	1	001
2	2	010
3	3	011
4	4	100
5	5	101
6	6	110
7	7	111
8	10	1 000
9	11	1 001
10	12	1 010

Note the similarity between octal and decimal: they are the same from zero to 7 and differ only from eight up. The difference arises, because of the definition of the base of a number system. In a base-10 or decimal system, the largest number we can represent in one digit is 9, or one less than the base. In the base-8 or octal system, the largest number that we can represent is 7, which is again one less than the base. Consequently, the base is the first number in the system that requires two digits to represent. Thus, eight in the octal system is represented by 10, just as ten in the decimal system is represented by 10.

Conversion between octal and binary is far easier than between decimal and binary. Examination of the table above will show why, since each octal number between zero and seven can be represented by no more than three binary digits. As a result, conversion is simply a matter of dividing a binary number into groups of three digits starting at the right end, and writing down the octal equivalent of each group.

Thus 101011010011101 is divided into groups

101 011 010 011 101	and the conversion performed by simply
5 3 2 3 5	writing down each octal digit separately.

There is no reference made between binary groups, nor are there any carry-out calculations to be made.

Conversely, we can convert octal to binary by simply writing down the three-digit binary number equal to each octal digit. The octal number 12345 is converted as follows:

1 2 3 4 5

001 010 011 100 101

The octal-binary conversion process is so basic to small computer programming that it should be committed to memory as rapidly as possible.

3. Octal Arithmetic

The only rule necessary to perform addition in octal is 7 + 1 = 10. Remembering this single rule will automatically remind you that 7 + 2 = 11, and 6 + 4 = 12 and so forth. For example:

$$\begin{array}{r} 22 \\ +63 \\ \hline 105 \end{array} \qquad \begin{array}{r} 246 \\ +153 \\ \hline 421 \end{array}$$

Looking at the examples on page 3, we can simplify the binary addition problem by converting to octal.

$$\begin{array}{r} 10111 \\ +10001 \\ \hline \end{array} = \begin{array}{r} 10\ 111 \\ +10\ 001 \\ \hline \end{array} = \begin{array}{r} 010\ 111 \\ +010\ 001 \\ \hline \end{array} = \begin{array}{r} 2\ 7 \\ +2\ 1 \\ \hline 5\ 0 \end{array} = 101\ 000$$

More important, 20-bit numbers are reduced to more tractable form using octal, as shown in this second example from page 3:

$$\begin{array}{r} 01\ 110\ 111\ 010\ 110\ 111\ 011 \\ 01\ 101\ 101\ 101\ 010\ 100\ 110 \\ \hline \end{array} = \begin{array}{r} 1\ 6\ 7\ 2\ 6\ 7\ 3 \\ 1\ 5\ 5\ 5\ 2\ 4\ 6 \\ \hline 3\ 4\ 5\ 0\ 1\ 4\ 1 \end{array} = 11\ 100\ 101\ 000\ 001\ 100\ 001$$

At this point we can introduce an important rule of thumb that may be useful in performing addition in octal. Add each pair of digits in your head in decimal. If the sum is greater than seven, subtract eight. The remainder is the digit to be put down in that column, and one is the carry. For example:

$$\begin{array}{r} 7 \\ +5 \\ \hline (12_{10}) \end{array} \quad \text{then } 12_{10} - 8_{10} = 4. \quad \text{So} \quad \begin{array}{r} 7 \\ +5 \\ \hline 14_8 \end{array}$$

(The subscripts 8 and 10 refer to the number base used.)

4. Conversion Between Octal and Decimal

The conversion between octal and decimal is performed less often. Using positional notation for base 8, the number 246_8 means

$$\begin{array}{rcrcr} 2 \times 8^2 & = & 2 \times 64 & = & 128 \\ +4 \times 8^1 & = & 4 \times 8 & = & 32 \\ +6 \times 8^0 & = & 6 \times 1 & = & 6 \\ \hline & & & & 166_{10} \end{array}$$

However, one need not carry out this tedious operation since an octal to decimal conversion table is provided in Appendix V.

5. Conversion Between Decimal and Octal

For numbers within the range of the table in Appendix V, the easiest way to convert from decimal to octal is simply to look them up in the table. However, the general method for decimal-octal conversion is to subtract various powers of eight from the decimal number, recording the number of subtractions per power as the corresponding octal digit.

To convert the decimal number 2453 to octal, we begin by subtracting $8^3 = 512$:

	2453	1941	1429	917
	- 512	- 512	- 512	-512
	1941	1429	917	405
number of subtractions:	1	2	3	4

The octal digit for the 8^3 column is therefore 4.

We then proceed to subtract 8^2, or 64. There are so many subtractions here, however, that division becomes easier, and so we divide 405 by 64:

$$\frac{405}{64} = 6, \text{ with a remainder of } 21$$

The digit for the 8^2 column is therefore 6. We then divide by 8^1, or 8, and get

$$\frac{21}{8} = 2 \text{ with a remainder of } 5$$

The 8^1 column digit is thus 2, and the 8^0 column digit = 5, since

$$\frac{5}{8^0} = \frac{5}{1} = 5$$

The converted number, then, is

$$2453_{10} = 4625_8.$$

While this method is unnecessary for smaller numbers available in most tables, for larger numbers, subtraction of or division by various powers of eight is useful until the remainder is in the range shown by the table.

6. <u>Exercises</u>

(1) Convert the following binary numbers to octal:

010	010100101011110111000
101101	100101111101000110110
00010101	100101011011110001010
1011010101	

(2) Convert the following octal numbers to binary:

223	1264
11707	65643
2106463	3006557

(3) Convert the following octal numbers to decimal:

7777	10000
144	256
4076	12346

(4) Convert the following decimal numbers to octal:

4096	524,289
100	16,383
512	
300	

(5) Perform the following octal additions:

```
 2 4 6 7     1 2     1 0 5 4 3     3 0 4 5 6 6
 1 2 3 4     2 3     2 1 6 1 5     1 3 4 6 5 2
--------    ----    ----------    ------------
```

(6) Perform the following binary additions directly and by conversion to octal. Compare your results.

```
 110 101 111 001 100
 001 010 000 111 101
--------------------

 10 100 001 101 111 101 100
 01 111 010 001 011 111 011
---------------------------
```

D. Important Numerical Concepts for 1080 Programming

1. Introduction

The minicomputer when first manufactured "knows" absolutely nothing. It understands no languages, nor Teletype commands. Since the machine is completely empty when it is built, it is only fitting that the programmer go halfway in learning to talk to it in concepts it can understand. These concepts include a few numerical ones which may be unfamiliar to the average scientist.

2. Complements

The term complement, or more fully, one's complement is extremely useful in referring to a binary machine. Put most simply, the one's complement of a number is obtained by changing all zeros to ones and all ones to zeros. Thus 000 and 111 are complements. Similarly 101 and 010 are complements.

The binary number

01 101 010 101 111 000 100 is the complement of
10 010 101 010 000 111 011, and vice-versa.

Looking at it another way, the sum of two binary numbers which are complements must be all ones since a one must always line up with a zero during addition. Thus the following complements sum to produce all ones:

```
000    101    010 101 000 110
111    010    101 010 111 001
---    ---    ---------------
111    111    111 111 111 111
```

This second approach leads to the suggestion of a method for determining the complements of <u>octal</u> numbers without converting them to binary. Since the sum of two binary complements must be all ones, the sum of two octal numbers which are complements must be all sevens. Remember that $111_2 = 7_8$.

It is obvious, then, that we can determine the complement of an octal number by simply subtracting it from the octal number representing a binary number which is all ones.

The complement of 101 100 010 can be determined as follows:

```
101 100 010 = 5 4 2         7 7 7
                           -5 4 2
                           ------
                            2 3 5 = 010 011 101
```

In the case of a twenty bit number, the complement must be determined by subtracting it from that number which represents all twenty bits equal to 1. This number is 11 111 111 111 111 111 111 or 3 7 7 7 7 7 7. The first digit is only a three because the two left-most bits are left over after dividing the twenty bits into groups of three starting at the right. Thus, the twenty bit complement of 456 is determined by subtracting 456 from 3777777.

```
 3 7 7 7 7 7 7                   3 7 7 7 7 7 7
-        4 5 6                  -1 2 5 6 0 0 0
--------------                  --------------
 3 7 7 7 3 2 1      Similarly,   2 5 2 1 7 7 7
```

3. <u>Two's Complement</u>

The two's complement is closely related to the one's complement and is of particular use in the 1080. The two's complement of a binary number is simply defined as the <u>one's complement plus one</u>.

The two's complement of 1 101 is 0 010 + 1 = 0 011.

```
The two's complement of 1 2 3₈ =  7 7 7
                                 -1 2 3
                                 ------
                                  6 5 4 + 1 = 655₈.
```

Using 20-bit numbers, the two's complement of 5 3 2 6 is found by

```
  3 7 7 7 7 7 7
-       5 3 2 6
---------------
  3 7 7 2 4 5 1 + 1 = 3772452₈.
```

It is also possible to perform this conversion in one step. Instead of subtracting the number from 3777777, we can subtract it from 3777777+1, which we will write in whatever form is most useful for this subtraction. To form the two's complement of 1256 we subtract 377777"8".

```
 377777"8"
-   125 6
----------
 377652 2
```

To form the two's complement of 123560, we subtract 37777"8"0.

```
 37777"8"0
- 1235 6 0
----------
 36542 2 0
```

Two's complement arithmetic is of great importance because the 1080, as well as a number of other minicomputers, utilizes the two's complement of a number as its negative. The total range of unsigned numbers that can be represented in the 1080 is 0 - 3777777_8. Arbitrarily, half of all these numbers are called positive and their two's complements are then called negative. The range of signed numbers then looks like this:

```
                          0
|-------------------------|-------------------------|
 2000000        3777777      1                1777777

       Negative numbers          Positive numbers
```

Close examination of those numbers in the negative range reveals that they all have one thing in common: the leftmost bit, bit 19, is set to one. Conversely, all positive numbers, including zero, have bit 19 set to zero. Consequently, bit 19 is called the sign bit and can be independently tested to allow a decision on whether a particular number is negative.

This division of numbers into positive and negative ranges is somewhat less arbitrary than it first appears since arithmetic manipulations can be performed in a consistent manner. Let us suppose that we start adding ones to some large binary number. The following will happen:

```
 3 7 7 7 7 7 5
 3 7 7 7 7 7 6
 3 7 7 7 7 7 7
*0 0 0 0 0 0 0    *At this point overflow occurs and there is a carry
 0 0 0 0 0 0 1     to the 21st bit.
 0 0 0 0 0 0 2
 0 0 0 0 0 0 3
```

The same sort of thing would happen if we started adding ones to -3. The sequence would be -3, -2, -1, 0, 1, 2, 3. In fact, as far as the computer is concerned this is what we have done. The two's complement of 3 is 3777775_8 and thus represents -3 to the computer.

As an additional check we add 3777775 (or -3) to +3:

$$\begin{array}{r} 3\,7\,7\,7\,7\,7\,5 \\ 3 \\ \hline (1)\ 0\,0\,0\,0\,0\,0\,0 \end{array}$$

and get zero as we expect. The 21st bit, if it existed, would be set by this operation as indicated by the (1).

4. Subtraction

The 1080 performs subtraction in the same manner as we are taught to do in algebra: by changing the sign and then adding. The sign is changed in the computer by negating or, in other words, by taking the two's complement of the number. Thus, if we wished the computer to subtract 5 from 7 it would proceed as follows:

$$\begin{array}{rcr} +7 & = & 7 \\ -5 & = & \underline{3777773} \\ & & 0000002 \end{array}$$

In other words 7 - 5 = 2

5. The Logical And

One operation that computers can perform easily that is not generally performed in arithmetic is the logical AND. The result of a logical AND between two bits is found as follows:

a. If both bits are ones, the result is one.
b. If both bits are zeros, the result is zero.
c. If the two bits are different, the result is zero.

This is represented in a truth table below:

	0	1
0	0	0
1	0	1

The AND function can be thought of as a masking operation between two numbers. What you want to examine remains the same; what you are not interested in examining, becomes 0. If you wish to examine only bit 12 of a number, perform a logical AND between that number and another number having only bit 12 set. Remembering that we number bits from the right starting with 0, that mask would be 0010000_8.

Thus, the logical AND between 3456732 and 0010000 produces:

3 4 5 6 7 3 2		11 100 101 110 111 011 010
0 0 1 0 0 0 0		00 000 001 000 000 000 000
0 0 1 0 0 0 0	or, in binary	00 000 001 000 000 000 000

If we wish to examine only one octal digit, we need only AND that digit with all ones, and AND all other digits with zeros. ANDing one octal digit with ones is the same as ANDing it with a 7. To examine the fourth octal digit of a twenty bit octal number, we perform the following operation:

3 4 5 6 7 3 2		11 100 101 110 111 011 010
0 0 0 7 0 0 0		00 000 000 111 000 000 000
0 0 0 6 0 0 0	or, in binary	00 000 000 110 000 000 000

6. The Inclusive OR

Another function easily performed electronically by a digital computer is the inclusive OR. For this operation, the following rules apply for the ORing of two bits.

a. If both bits are zero, the result is zero.
b. If the bits differ, the result is one.
c. If the bits are both ones, the result is one.

Put in truth table form, the inclusive OR is represented as follows:

	0	1
0	0	1
1	1	1

The inclusive OR is used to find out whether bits are turned on in either or both of two computer words. In other words, the result shows which bits have ones in common.

The inclusive OR between octal digits is shown below:

000	0	010	2	100	4	101	5
111	7	110	6	011	3	001	1
111	7	110	6	111	7	101	5

7. Exercises

(1) Find the one's complement of the following numbers:

01 110 111 011 101 110 001   00 111 101 010 110 100 100

00 000 000 000 101 111 000   10 111 101 101 111 010 011

(2) Find the two's complement of the above numbers.

(3) Perform the following subtractions using two's complement octal arithmetic. Assume twenty bit results.

5 6 3 4 2	1 0 6 7 5 4 2	1 1 0 5 4 2 3
- 3 1 5	-2 1 3 4 5 2 7	-3 2 5 4 3 2 1

(4) Write the positive octal number corresponding to each of the following negative octal numbers:

3777560   2456120   3453210

(5) Perform the AND operation between these octal numbers:

7	5	3 7 6	3 7 7 0 7 7 0
3	2	1 2 3	1 2 4 6 2 3 7

(6) Perform the inclusive OR operation between the above octal numbers.

II. BASIC PROGRAMMING CONCEPTS

A. Addresses

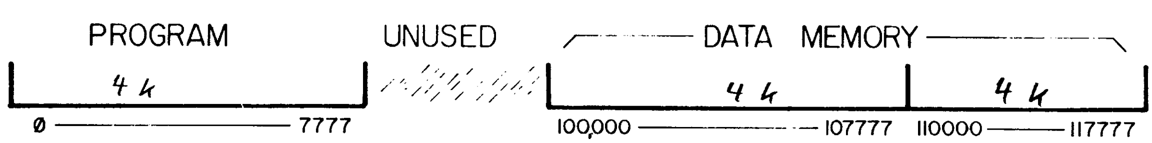
Each 20-bit word in the 1080 has associated with it both an address and contents. The contents are the actual configuration of the bits in that word and the address is its sequential location in memory. The addresses are simply numbers describing this location, beginning with address zero. Since the computer deals with addresses in binary form, it is convenient to represent them as octal numbers just as the contents of words are represented in octal.

For mechanical reasons, memory is provided in sections of 4096_{10} words (or 4K) called stacks. The typical 1080 system consists of one stack to be used for program storage and one or more stacks to be used for the storage of signal averaged data. These two sections are referred to as program memory and data memory respectively. This distinction is quite arbitrary and does not affect the amount of memory that can be allocated for either purpose in any way.

The addressing of the first 4K stack, usually utilized for program memory, begins at address Ø and is numbered sequentially through 7777_8. This comprises 10000_8 words of storage or 4096_{10}. Additional stacks set aside specifically for programming would begin at 10000 and run through 17777, at 20000 through 27777 and so forth.

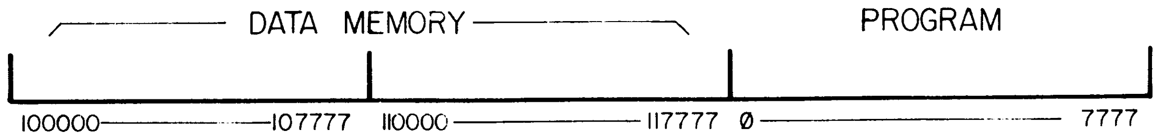
The section of memory set aside for data accumulation under wired program control begins at address $100{,}000_8$ and proceeds through 107777 in the first 4K. Each additional stack is addressed sequentially from there. The only difference in memory set aside for data accumulation is that signal averaging and display automatically begin at address 10,0000 if the Readout or Measure Memory Allocation switches are set to Starting = Ø.

In a 12K machine, the memory layout looks like this:



The program memory can be used for data storage however, so that data can be signal averaged into all stacks. The program stack is utilized during data acquisition or readout if the size of memory selected is greater than the number of stacks in data memory. In this case, the Program Protect pushbutton must be out. If this button is depressed, the first 4K of program memory is protected from destruction by the wired processor. It is never protected from access by the stored program processor.

During data acquisition, the layout of memory appears to be as shown below to the wired processor:



The program memory section would be accessed only if more than 8K was selected as the Measure Memory Allocation Size.

While the stored program processor generally runs programs located in the first 4K, it is not restricted to this section of memory. The processor can be started at any existing address and will automatically execute instructions sequentially from that point. Generally, however, 4K is sufficient for all data reduction and theoretical programs, and the remaining memory can be utilized for data storage. The number of memory stacks allocated to program (addresses below 100000_8) vs. the number allocated to data (addresses above 100000_8) can be manually adjusted by a switch setting within the 1080.

B. Registers Used in the 1080

1. The Instruction Register

As mentioned above, the programs which the stored program processor executes are simply sequences of binary numbers stored in memory. They are completely indistinguishable from signal averaged data. They differ only in how they are interpreted. A binary number can be called into the arithmetic unit and added to another piece of digital data from the analog-to-digital converter and the sum stored in memory. In this case, the number is a data point in some spectrum.

On the other hand, the number could be brought from memory into the Instruction Register and interpreted as a command to perform one or more elementary logical operations which comprise the 1080's instruction set. The actual operations performed are determined by which bits are set in a particular instruction word. For instance, let us consider an elementary instruction register only 4 bits long.

FETCH	ADD	SUB	STORE

Upon starting, the stored processor is given some initial address from which to retrieve the first instruction. This instruction is brought from core memory into the instruction register and interpreted. In our elementary

Instruction Register, there are four possible operations to be performed: getting data, addition of data, subtraction of data, and storing of data. Which of these operations is actually performed is dependent on the bits set. If the instruction was 1000_2 this would instruct the processor to fetch data from memory. If an addition operation were to be performed, the instruction would be 0100, and if both a fetch and an add were to be performed the register would contain 1100.

The 1080 Instruction Register is tied to 20 lights in the middle row of the Display Control section of the 1080. It can be observed while the processor is running.

2. <u>The Program Counter</u>

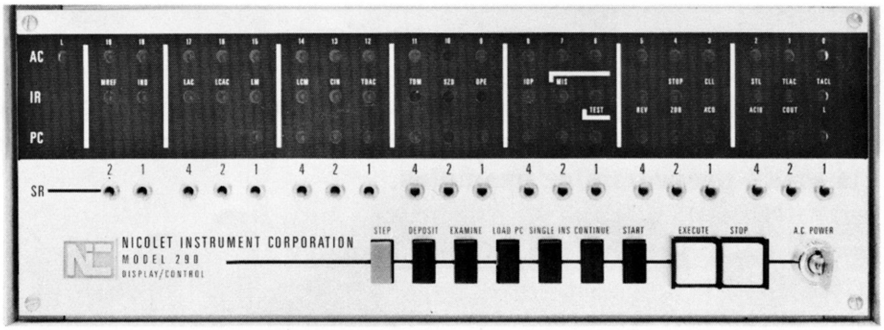
After each instruction is performed, some part of the stored processor must specify the address from which the next instruction is to be retrieved. This register is called the Program Counter. It always contains the address of the instruction to be executed <u>after</u> the current one. The Program Counter is tied to the bottom row of lights.

3. <u>Accumulator</u>

The accumulator is the one register that can be manipulated by the programmer. Numbers can be added or subtracted and examined there. Multiplication and division by powers of two also take place there as do logical ANDs and ORs. Numbers are fetched from memory and displayed there for various purposes and numbers in the accumulator can be stored in locations in memory.

The accumulator consists of 20 bits and a 1-bit extension called the <u>link</u>. If an addition is performed in the accumulator and the result requires more than 20 bits to represent, the overflow will be found in the link.

The accumulator and link are tied to 21 lights on the top row of the Display Control section of the 1080. They can be observed while the processor is running. The accumulator (AC) link, instruction register and program counter are pictured below.



4. The Multiplier-Quotient Register

The multiplier-quotient register, or MQ, is used in multiplication, division, bit inversion and logical OR operations. Since it is not tied to a row of lights it can only be examined by transferring its contents to the AC.

5. The Zero Test Register

The Zero Text Register is a piece of logic that tests a number for zero. If the number is not zero nothing happens. If the number is zero the program counter (PC) is incremented by 2 instead of by 1. This causes the processor to skip the next instruction in sequence. In other words, if the zero test register detects a zero, the next instruction is skipped.

C. Programming the 1080 Using Group I Instructions

1. Mnemonics

All of the arithmetic operations that the 1080 can carry out are performed in a set of instructions called Group I Instructions. These instructions involve addition, complementing, and incrementing as well as transfers between the accumulator, memory and the Zero Test Register.

The only meaningful instruction to the computer is the combination of ones and zeros loaded into the instruction register. These combinations are so abstract, however, even when abbreviated in octal, that they are difficult to remember and apply directly. For this reason, it is convenient to develop a series of abbreviations for each instruction which remind us of their actual function. These abbreviations are called mnemonic codes. They have no direct meaning to the computer, but each mnemonic has a corresponding binary equivalent which is meaningful to the computer. We will see in Chapter V that the Assembler computer program translates these codes into their binary equivalents, freeing the user from ever having to know them.

2. Subgroups of Group I Instructions

We will first indicate the actual operations which the programmer can perform using Group I instructions, and will subsequently show the value of each of these in small programming examples.

a. Loading the AC and Memory

The two registers AC and memory, where memory means any memory location, can be examined and its contents transferred elsewhere using the two instructions

ACC	accumulator
MEM	memory

b. Addition and Subtraction

The 1080 can perform addition or subtraction between the accumulator (AC) and memory. In each of these instructions, A represents the AC and M represents memory.

A+M	accumulator plus memory
A-M	accumulator minus memory
M-A	memory minus accumulator

c. Incrementing and Decrementing

Both the AC and memory can be incremented or decremented using these instructions:

APO	accumulator plus one
MPO	memory plus one
AMO	accumulator minus one
MMO	memory minus one
AMP	accumulator plus memory plus one

d. Complementing

The one's complement of memory or the AC is taken as follows:

ACP	complement of the accumulator
MCP	complement of memory
ACM ~~ACP~~	accumulator plus the complement of memory
CAM	complement of the accumulator plus memory

e. Negation

Numbers in the AC or memory can also be negated. Remember that negation means taking the two's complement, or taking the one's complement and incrementing.

ANG	negative of the accumulator
MNG	negative of memory

f. Constants

Several constants can also be created for direct use in programming. This saves the necessity of storing the constants in some memory location.

ZER	zero
ONE	one
MON	minus one
MTO	minus two

g. Logical AND

The AND instruction performs a logical AND between a memory location and the accumulator. Thus, there are always two operands regardless of the fact that the mnemonic itself does not necessarily imply them. The AND produces a 1 if and only if both operands have a 1 in that bit position and a zero otherwise. It is most commonly used to "mask" particular bits of a word and examine them separately. For example to examine bits 0-2 of a word containing 3765745, we would AND that word with 0000007, giving 0000005, or the contents of bits 0-2, with all other bits set to Ø.

3. Destinations

All of the above instructions specify a source, either the AC or memory or both, but do not specify the destination, or the place in which the result is to be put. There are three possible destinations specified in the 1080, represented by the three suffixes A, M and Z, meaning accumulator (A), memory (M), and zero test register (Z). They may be specified in any combination and in any order.

For instance, to add the contents of the accumulator to a memory location we simply write

A+MM

This instruction is read "accumulator plus memory to memory." In this case the accumulator is unchanged, but the sum is stored in memory. We could perform the same addition leaving the memory location intact by writing

A+MA

which simply means accumulator plus memory to accumulator. The AC changes but memory remains unchanged. We could perform this addition without changing either register if we simply wished to test the result for zero

A+MZ.

This is read "accumulator plus memory to zero test register," or add the AC and memory <u>and skip if zero</u>. In this case, the two numbers are summed and if their result is zero, the next instruction is skipped. Note that in every case only the destination register is changed. The source register is unmodified. Thus

ACCM

places the contents of the AC into memory. Memory is changed, the AC is <u>not</u>.

4. <u>Syntax</u>

A Group I Instruction can thus be divided into two sections: a source and a destination. These two sections can also be referred to as an <u>operator</u> and a <u>suffix</u>, where the operator is one of the three-character codes given above, and the suffix is one or more of the destinations A, M and Z. The suffixes can be given in any order. For example

A+MMZ is the same as A+MZM

and both mean add the AC to memory, store the result in memory, and skip if the result is zero. Note that there is <u>no space</u> between the operator and the suffixes. While this is a minor distinction now, it will become more important when we discuss the Assembler program for translating the mnemonics into binary code.

D. <u>Addressing Modes</u>

1. <u>Bit Assignments</u>

The actual bit assignments in the Instruction Register for Group I instructions are as follows:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							A	M	Z										
ADDRESS MODE		OPERATOR					SUFFIX			OPERAND (ADDRESS)									

Bits 13 through 17 specify which Group I instruction, and bits 10 to 12 specify which of the three suffixes are used. Up to this point we have not discussed how a particular memory location is accessed as data. This is accomplished using bits 0 - 9 to represent the data and bits 18 and 19 to represent the addressing mode. There are three such addressing modes, **immediate**, **direct**, and **indirect**.

2. The Immediate Mode

In the immediate addressing mode the actual instruction contains the data. Bits 0 - 9 contain the actual binary number operated upon. Bits 18 and 19 both contain zeros to indicate the immediate addressing mode:

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Ø	Ø						A	Ø	Z										
IMMEDIATE MODE		GROUP I INSTRUCTION					SUFFIXES A AND Z ONLY			DATA									

Since the instruction is the data the M suffix or destination cannot be used. If it were this would imply that an instruction could change itself. This is not possible in the 1080.

The symbol we will use to specify the immediate mode is the left parenthesis (. We can place any number to the right of the parenthesis that can be represented in the 10 data bits. This number range is between Ø and 1777_8. For instance, to place 230_8 in the AC we give the computer the instruction

```
MEMA (230    /LOAD 230 INTO THE ACCUMULATOR
```

This instruction means "take the number in the right hand ten bits of the instruction and load it into the accumulator." The data location is the right hand half of the actual instruction word. The comment following the slash (/) is purely to remind us what operation we are performing. It is not interpreted by the computer in any other way.

To add 15_8 to the accumulator we write

```
A+MA  (15    /ACCUMULATOR PLUS MEMORY TO ACCUMULATOR
```

If we performed the above two instructions sequentially the AC would contain 230 at the end of the first instruction and 245 at the end of the second instruction. All of these numbers are, of course, in octal.

In the immediate mode, any of the Group I instructions can be performed, to extend the range of the numbers which can be represented. While bits 0 - 9 can only represent numbers from 0 - 1777, negative numbers can be created by such instructions as

```
MNGA  (115   /NEGATIVE OF MEMORY TO ACCUMULATOR
```

which means place the negative of 115 into the AC, or place 3777663 in the AC. Similarly, subtraction can be performed in the immediate mode.

```
MEMA  (1015  /MEMORY TO ACCUMULATOR
A-MA  (230   /ACCUMULATOR MINUS MEMORY TO ACCUMULATOR
```

This example causes 1015 to be loaded into the AC in the first instruction and 230 to be subtracted from it in the second. The AC then contains 565.

The following instruction allows the programmer to test the AC for any number accessible in the immediate mode:

```
A-MZ  (215     /SKIP IF THE AC IS 215
```

A skip is performed if and only if the AC is 215; that is, if the AC minus 215 equals zero. This is the first elementary decision that the 1080 can make.

The range of numbers that can be represented in the immediate mode is from -2000 to $+2000_8$. The extremes are shown below:

```
MPOA  (1777     /SET THE AC = 1777+1, OR 2000
MCPA  (1777     /SET THE AC TO THE COMPLEMENT OF 1777
```

The complement of 1777 is found by

```
  3 7 7 7 7 7 7
-       1 7 7 7
  -------------
  3 7 7 6 0 0 0
```

The number 3776000 can be shown to be the two's complement of 2000 or the negative of 2000 by

```
    3 7 7 6 0 0 0
+         2 0 0 0
    -------------
(1) 0 0 0 0 0 0 0   where the (1) is the carry.
```

3. <u>The Direct Mode</u>

In the direct addressing mode, the right hand ten bits of the instruction refer to an <u>address</u> from which the data is taken. The direct mode is symbolized by one or more spaces between the last suffix and the beginning of the address istelf. In this mode bit 19 is turned on indicating that the instruction references memory.

19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	Ø						A	M	Z										
DIRECT MODE		GROUP I INSTRUCTION					SUFFIX			ADDRESS OF DATA									

```
MEMA 123
```

means get the contents of address 123 and place that in the accumulator. In this mode, the user is no longer limited to 10 bit numbers or their complements, since the entire 20-bit contents of that address is moved.

Other examples of the direct mode include

```
ACCM 406     /PLACE THE CONTENTS OF THE AC IN ADDRESS 406
A+MM 201     /ADD THE AC TO MEMORY LOCATION 201
MNGA 372     /PLACE THE NEGATIVE OF ADDRESS 372 IN AC
A-MMZ 514    /SUBTRACT THE CONTENTS OF LOCATION 514 FROM THE AC
             /PLACE THE RESULT IN LOCATION 514 AND SKIP IF ZERO
```

With this information, we can now write a simple program to add three numbers together. We will assume that these numbers are already stored in locations 100 - 102, and that the result of the addition should appear in the AC when the addition is complete. As is good programming practice throughout, we will comment each line to describe exactly what operation is being performed.

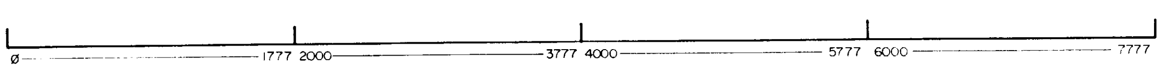
```
*Ø     /SYMBOL TO INDICATE THE STARTING ADDRESS FOR THIS CODE

MEMA 100     /LOAD THE FIRST NUMBER INTO THE AC
A+MA 101     /ADD THE SECOND NUMBER TO IT
A+MA 102     /ADD THE THIRD NUMBER TO THAT
STOP         /AND HALT THE PROCESSOR WITH SUM IN AC
```

The program starts by loading the contents of address 100 into the AC. This is a jam transfer; the previous contents of the AC are lost. The contents of address 100 are unaffected. The program next adds to that number the contents of address 101. It does the same thing again, adding the contents of address 102 to that sum and then stops interpreting instruction at the STOP command. The results of the addition are left in the AC and can be read from the lights on the front panel.

4. <u>Paging</u>

Obviously, the ten bits representing the address can only represent addresses from Ø to 1777, although there are far more memory locations in the machine. In order to conquer this problem, each memory stack has been divided into four <u>pages</u> of 1024_{10} or 2000_8 words each. They are laid out as follows:



The right hand ten bits of the instruction is then considered the <u>page relative</u> address, so that MEMA 123 accesses address 123 if on page Ø, address 2123 if on page 2000, address 4123 if on page 4000, address 6123 if on page 6000 and so forth. The address of the beginning of the page is added to the relative address by the hardware and that address is then accessed by the computer. Consequently one can address any of 1023_{10} other memory locations directly from any given place in memory.

5. Indirect Mode

Obviously a computer that could only address 1024 locations by any means would be inadequate for scientific purposes, where data arrays may be as large as 32,768 points. A third mode of addressing is provided in which a full word points to the actual address desired. In this mode, called indirect addressing, a memory location on the same memory page as the instruction contains the address of the word actually to be accessed. The "at" sign (@) is used to indicate indirect addressing.

Address	Mnemonic	
200	MEMA @ 436	/GET THE CONTENTS OF 5370
201	ACCM @ 437	/AND STORE IT IN 6120
202	STOP	/THEN HALT
436	5370	
437	6120	

The above program is read "Get the contents of the address pointed to by address 436 and then store this number in the address pointed to by address 437." The result is that the contents of address 5370 are loaded into the AC from an instruction located on another memory page and this number is then deposited in address 6120.

The usual use for the indirect mode is in accessing arrays of data stored in the data section of memory. It is not always necessary to reserve a pointer for each element in such a list, however, since one can set a pointer to the top of the list and then advance the pointer from one element to the next in a loop. The bit assignments for indirect addressing are the same as for direct addressing except that bit 18, the indirect bit, is one.

To access only two or three locations in data memory, one can operate through a set of pointers. The following program adds the first and last points of a 4K data array and stores the result in the first point. Remember that data memory begins at address 100000_8.

```
/PROGRAM TO ADD THE FIRST AND LAST POINTS OF DATA MEMORY
*Ø              /STARTING ADDRESS OF THE PROGRAM

MEMA @ 10       /GET THE LAST DATA POINT INDIRECTLY
A+MMA @ 11      /ADD IT TO THE FIRST, STORE AND
STOP            /STOP WITH RESULT IN AC AND IN MEMORY

*1Ø             /POINTERS STORED AT ADDRESS 1Ø AND 11
107777          /ADDRESS OF LAST POINT IN 4K ARRAY
100000          /ADDRESS OF FIRST POINT IN ARRAY
```

This program accesses the contents of address 107777 and places it in the AC. This number is then added to the contents of address 100000 and the result stored in location 100000. The sum is also placed in the AC, since both the A and M suffixes are used. The sum can then be read from the AC lights when the program stops.

6. Exercises

(1) Explain why the M suffix cannot be used in the immediate addressing mode.

(2) The second program shown in section D-5 above was loaded into the computer and executed. Despite the fact that it occupied only locations 0, 1 and 2, the Program Counter showed a value of 3 when the program halted. Explain this.

(3) What will the AC be at the end of this program?

```
MEMA (123
MNGA (456
A-MA (3
STOP
```

(4) What would the configuration of a 1080 have to be for the following instruction to serve a useful purpose?

```
MPOM @ 113
*113
10000
```

III. PROGRAMMING THE 1080 IN ASSEMBLY LANGUAGE

A. Labels

By the end of the previous chapter, it became apparent that it would be extremely difficult to keep track of the addresses of every instruction and constant used in a relatively lengthy program so that they could be accurately addressed. It is possible, however, to represent the actual addresses symbolically and let the Assembler program take care of the translation not only of the instructions into binary equivalents, but also take care of the translation of symbolic addresses into their binary equivalents.

The use of labels for various addresses simplifies the task of the programmer in keeping track of memory address allocation. A label is a name given an instruction or a constant so that it can be referred to in the program. The following rules apply to such labels:

(1) The name may be between 1 and 6 characters long. Labels with differing characters beyond the sixth are considered identical.

(2) The first character must be an alphabetic one.

(3) The label must not contain any embedded blanks.

(4) An address is defined as labeled by giving it a name followed by a comma.

(5) A label may not contain a dollar sign.

The following example is given using both absolute addressing and labeled addressing. The Assembler program would treat these as equivalent.

	*Ø	Absolute Address	*Ø
START,	MEMA LABEL1	Ø	MEMA 4
	A+MA @ LABEL2	1	A+MA @ 5
	ACCM TEMP	2	ACCM 6
	STOP	3	STOP
LABEL1,	2123004	4	2123004
LABEL2,	4230	5	4230
TEMP,	Ø	6	Ø

The four labels used in the above program are START, LABEL1, LABEL2 and TEMP. Only the last three of these are referred to by the program.

This program loads the contents of location 4 into the AC, adds to it the contents of location 4230 (by indirect addressing) and stores the result in location 6. Note that there is no need to keep track of addresses if labels are used to refer to the address of memory locations.

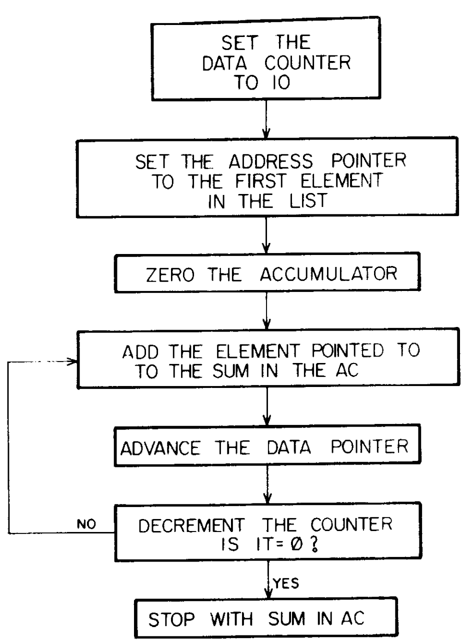
Here it should be emphasized again, that the mnemonics for various instructions are meaningless to the computer per se. They are interpreted by a computer program called the Assembler into binary numbers and addresses. This translation simply is a set of table look-ups in which the instruction MEMA 123 is decomposed into MEM, A and 123. The value for each of these components is looked up and the result combined to give the final binary instruction.

B. A Program to Add 10 Numbers Together

In the case of the addition of two or three numbers, it is not unreasonable to address each of them individually. There is little or no efficiency to be realized in any more elaborate addressing scheme. However, if we wish to sum 10 or more numbers together, it is desirable to use the same pointers over again. In writing any program it is desirable to follow the general outline given below:

(1) Define the task clearly in good grammatical sentences.
(2) Draw a flowchart of the program's logic.
(3) Write the program code, commenting it thoroughly.
(4) Test and debug the program.

We will state the problem as follows. This is a program to add ten numbers together that are already stored in memory starting at address 100000. Indirect addressing will be used to advance a pointer down a list. A flowchart of the program might look like this:



The program to accomplish this task, properly commented, is given below:

```
/PROGRAM TO ADD TEN NUMBERS TOGETHER

          *Ø                   /STARTING ADDRESS ZERO
START,    MEMA (12             /SET 1Ø (BASE-1Ø) INTO COUNTER
          ACCM COUNT
          MEMA PNTSET          /SET THE DATA POINTER
          ACCM POINT
          ZERA                 /SET AC = Ø
LOOP,     A+MA @ POINT         /ADD EACH DATA POINT INTO AC
          MPOM POINT           /ADVANCE THE POINTER
          MMOMZ COUNT          /ARE ALL 1Ø DONE?
          JMP LOOP             /NO, DO ANOTHER POINT, JUMP BACK TO LOOP
          STOP                 /YES, HALT WITH RESULT IN AC
COUNT,    Ø                    /POINT COUNTER
PNTSET,   1ØØØØØ               /BEGINNING OF DATA
POINT,    Ø                    /VARIABLE POINTER
```

The above program introduces the instruction JMP or jump. This is simply an unconditional jump to the location specified. During a JMP instruction, the address specified is transferred to the program counter so that the next instruction to be executed is fetched from memory at the address specified by the jump. The JMP is a pseudo-Group I instruction. Both direct and indirect mode addressing are legal although the immediate mode is prohibited and is in fact meaningless.

This program utilizes a loop of logic to add all ten numbers together with the same pointer pointing sequentially to each element in the list. The program starts by initializing a counter to 12_8 (or 10_{10}) and setting the pointer POINT to address 100000. The AC is zeroed and the contents of address 100000 is added to the AC. Then POINT is incremented from 100000 to 100001 and the counter decremented from 12 to 11. The program returns to statement LOOP where the number is added into the AC which is pointed to by POINT. POINT now contains 100001 so that the second number in the list of numbers is thus added to the AC. Then the pointer is incremented to 100002 and the counter decremented from 11 to 10. This loop continues until all 10 numbers have been added in the AC indirectly. At this point the counter has been decremented to 1. When the program passes through MMOMZ COUNT after the last addition, the counter is decremented to Ø, and since the result is zero the instruction JMP LOOP is skipped. Instead, the instruction STOP is executed, halting the program with the sum in the AC.

The final step in the sequence of good programming practice, testing and debugging the program, is usually accomplished by loading the program into memory, executing it, and comparing a known result with that found by the program. More extensive debugging techniques will be discussed in Chapter VI.

C. Communicating with the Teletype

1. Introduction

The Teletype is the input/output (I/O) device normally used for reading in binary tapes, entering instructions and data. It consists of four separate elements: (a) the keyboard, (b) the printer, (c) the tape reader and (d) the tape punch.

During offline (LOCAL) operation, all four of these sections are mechanically linked, so that typing a character produces an impulse to the printer and punch mechanisms causing that character to be printed and (if the punch is turned on) punched. Similarly, reading a character into the tape reader causes it to be echoed by the printer and punch. However this information is not sent to the computer.

During on-line (LINE) operation, the computer is connected to the Teletype, the reader and the keyboard are logically equivalent and the printer and punch are equivalent. A computer command to read from the Teletype causes the keyboard-reader to be queried and a command to print on the Teletype printer will also cause tape to be punched if the punch is turned on.

However, it must be emphasized that there is no link whatever between the keyboard-reader and the printer-punch during on-line operation. In fact, the only way the Teletype can be made to influence computer operations is if the computer has been specifically programmed to read and print on the Teletype.

2. Switch Functions

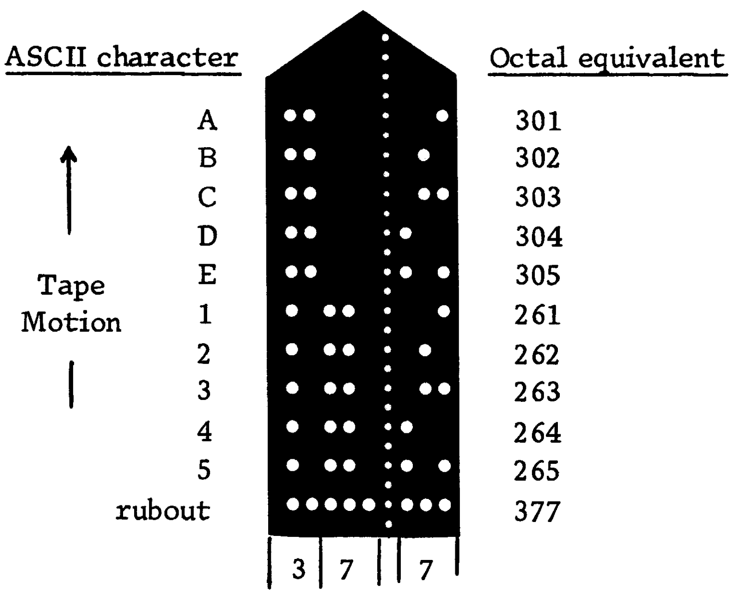
LINE-OFF-LOCAL: This switch is located below and to the right of the keyboard. It is the main power switch and controls whether the Teletype "talks" to the computer or only to itself. In the LINE mode, the Teletype sends out signals to the computer which it can recognize depending on how it has been programmed. In the LOCAL mode the Teletype acts just like a typewriter. It is wholly dissociated from the computer and can be used, for instance, to generate tapes while the computer is performing some other function.

START-STOP-FREE: This control on the tape reader controls tape motion. In the FREE position, the sprocket wheel moves freely. The reader should always be set to FREE during tape loading and unloading to prevent possible tearing of the tape. In the STOP position the sprocket wheel is immobile and in the START position, tape can be read by the computer.

Tape Punch Switches: If ON is depressed, the printing of any character will cause its duplication on the punch. Depressing REL (release) allows one to pull out old tape before loading in a new roll. B. SP. backspaces the tape one position each time it is firmly depressed. This is useful for correcting tapes prepared off-line.

3. Reading an ASCII Paper Tape

All characters punched by the Teletype are punched according to the ASCII code (American Standard Code for Information Interchange). Paper tapes punched contain eight rows of holes, representing binary numbers from 0 to 2^7-1, or octal numbers from 0 to 377_8. The binary to octal conversion is performed as usual, by grouping the binary bits into groups of three and converting each group to its octal equivalent. Hold the tape as shown below and use the conversion table at the right. Unlike tapes produced by actual computer programs, this one has had blank lines inserted between the punched ones to improve legibility for this example.



Examining the first line of the tape, the binary number 11 000 001 is found. Since $11_2 = 3_8$, $000_2 = 0_8$ and $001_2 = 1_8$, the number is 301_8. After consulting the ASCII character table in Appendix I one finds that this is the octal code for the letter A. Similarly, the sixth punched line contains the number 10 110 001 or 261_8. This is the ASCII code for the number 1.

Since all ASCII codes range between 200_8 and 377_8, a punched tape can always be recognized as ASCII rather than binary if its leftmost column, the 200 column, is punched.

4. Programming the Teletype

The 1080 computer operates at a rate of one memory cycle every two microseconds. It takes two such cycles to execute most instructions; instructions involving indirect addressing take three memory cycles. The Teletype, on the other hand, operates at a maximum rate of 10 characters per second, either sending or receiving. In order for the computer to communicate with the Teletype accurately, it is therefore necessary that it be slowed down to the speed of the Teletype. This is accomplished using a ready flag, a one bit register which indicates whether or not the Teletype is ready to transfer information. If the flag is set to one, data can be transferred, but if the flag is set to zero, the Teletype is not ready.

There are two sections to the Teletype in LINE mode, the keyboard-reader and the printer-punch. Each of these sections has a ready flag and a set of instructions interrogating that flag and directing transfer of data.

The keyboard-reader uses the following two instructions:

TTYRF	Skip if the reader is ready to transfer information
RDTTY	Read the keyboard-reader into the AC.

The instruction TTYRF tests the ready flag of the keyboard-reader and if the keyboard has been struck, or if there is tape in the reader and the reader is turned on, the next instruction is skipped. The instruction RDTTY reads the keyboard reader buffer into bits 0-7 of the AC. Each character on the keyboard is in ASCII code, eight bits long, but any combination of bits, whether ASCII or not, will be correctly transferred by the Teletype reader.

A typical routine for reading the Teletype would be

```
T1,  TTYRF     /WAIT FOR READY FLAG
     JMP T1    /KEYBOARD NOT STRUCK, JUMP BACK
     RDTTY     /READ KEYBOARD-READER INTO AC
```

This program waits in the two instruction loop TTYRF, JMP T1 until either the keyboard is struck or the tape reader is turned on with tape in it. When one of these conditions occurs, the ready flag goes up and a skip is performed, bypassing the instruction JMP T1. The instruction RDTTY is then executed and the character is transferred to the lowest 8 bits of the AC.

The printer-punch has an analogous set of instructions:

TTYPF	Skip when the printer-punch is ready
PRTTY	Print the character in bits 0-7 of the AC

The printer is considered ready when it is not printing a character.

Let us now consider a short routine to print the message NIC.

```
        MEMA (316   /LOAD THE ASCII CODE FOR "N" INTO THE AC
P1,     TTYPF       /WAIT FOR PRINTER READY
        JMP P1
        PRTTY       /PRINT THE N
        MEMA (311   /GET THE CHARACTER "I"
P2,     TTYPF
        JMP P2
        PRTTY       /PRINT THE I
        MEMA (303   /ASCII "C"
P3,     TTYPF
        JMP P3
        PRTTY       /PRINT THE C
        STOP
```

This program sequentially loads the ASCII codes for N, I and C into the printer buffer, waits for the printer to be ready and then prints each character. A typical routine for getting the Teletype to behave like a typewriter would be

```
T1,     TTYRF       /WAIT FOR KEYBOARD TO BE STRUCK
        JMP T1
        RDTTY       /READ CHARACTER INTO AC
P1,     TTYPF       /WAIT FOR PRINTER READY
        JMP P1
        PRTTY       /PRINT CHARACTER
        JMP T1      /GO BACK TO GET NEXT CHARACTER
```

The character remains in the AC after printing and can then be tested for some particular value if, for instance, certain characters are used as commands by the program.

When power is first applied to the Teletype, the reader flag can be in either state, and the reader buffer may well contain garbage. For this reason it is advisable to clear the reader buffer with a RDTTY command as the first instruction in any program that will use the Teletype.

D. The JMS Instruction

It becomes apparent that it would be eminently desirable to be able to recall certain sections of code without the necessity of rewriting them each time they are needed in the program. This is particularly useful in the case of such common routines as those controlling reading and printing from the Teletype.

This can be done by the jump to subroutine or JMS instruction. When a JMS instruction is executed, the following things occur:

(1) the address of the instruction following the JMS is deposited in the first memory location of the subroutine,
(2) execution of instructions commences at the second location of the subroutine.

In practice, this means that the computer keeps track of the location from which the subroutine was called so that the program can return to the location following the subroutine call by a JMP indirect instruction. Consider the following example, in which a routine to type out a single character is converted to the subroutine TYPE.

```
/ROUTINE TO TYPE OUT "NIC"

          *200

Address   Mnemonic
 200      MEMA (316   /PUT N IN AC
 201      JMS TYPE    /AND TYPE IT
 202      MEMA (311   /PUT I IN AC
 203      JMS TYPE    /TYPE IT
 204      MEMA (303   /PUT C IN AC
 205      JMS TYPE    /TYPE IT
 206      STOP        /AND HALT
 207    TYPE, Ø       /THIS LOCATION WILL CONTAIN RETURN ADDRESS
 210    P1, TTYPF     /WAIT FOR PRINTER READY
 211      JMP P1
 212      PRTTY       /PRINT CHARACTER IN AC
 213      JMP @ TYPE  /AND EXIT TO LOCATION POINTED TO BY TYPE
```

This routine, when started at location 200, loads the value 316 into the AC. It then executes a JMS instruction at location 201. The effect of this instruction is to put the address of the instruction following the call into the first location of subroutine TYPE. In this case, the address 202 is placed into location 207. The TYPE subroutine is then executed in the usual way. At location 213 the instruction JMP @ TYPE causes the program to jump to the location pointed to by TYPE, or location 202. Address 202 is therefore the next instruction executed.

Address 202 contains the instruction setting the AC to 311. The subroutine TYPE is called again, this time placing the value 204 into location TYPE. Exit from TYPE causes a jump indirectly to location 204 where the value 303 is set into the AC. Finally the JMS TYPE at location 205 causes the value 206 to be stored in memory location 207 (TYPE) and exit from TYPE causes the program to halt at location 206. Thus, we have shown a subroutine that can be called from anywhere in memory and from which correct exit is always assured. Subroutines can be called either directly or indirectly, in either case the address following the actual JMS is placed in the first location of the subroutine. It is important to remember that the first location of a subroutine is destroyed by the JMS itself. For this reason the first location is generally written as a zero when the program is coded.

E. Shift Instructions

There are three kinds of shifts possible with the 1080: logical, arithmetic and integer. The number of places shifted is controlled by an integer following the instruction, which can vary from 0 to 17_8. A logical shift is an end-around shift of the bits of the accumulator, so that the instruction RLSH 1 causes all bits to move one place right, and bit 0 to move around to bit 19. The direction is simply reversed by a left logical shift (LLSH).

The arithmetic shift is signed shift. It causes the sign bit to be propagated to the right during right shifts. During left arithmetic shifts, bits are dropped off the left end. Thus, if bit 19 (the sign bit) is 1, indicating a negative number, the instruction RASH 3 will cause bits 0 - 18 to be shifted three places right. Bits 0 - 2 will be lost, bit 18 will have moved to bit 15 and so forth. The sign bit will be copied into bits 18 - 16, making bits 19 - 16 all ones. This is illustrated below:

AC before instruction   10 001 010 110 011 100 101   (2126345 octal)

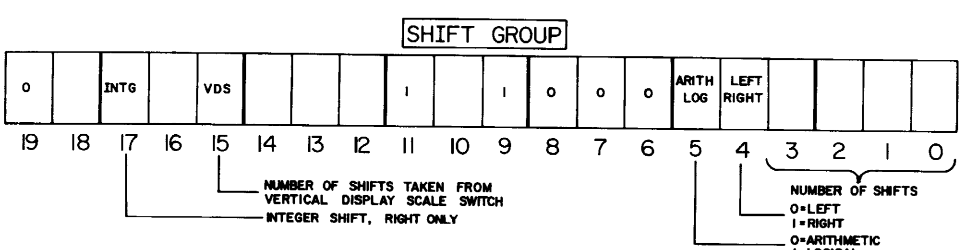
RASH 3

AC after instruction   11 110 001 010 110 011 100 (101 lost) 3612634_8

The RISH instruction causes the bits of the AC to be shifted right without regard to sign, with the least significant bits "falling off" the end. While LISH does not exist, LASH has the same effect.

The number of shifts performed in one instruction can vary from 0 to 15_{10} or 0 to 17_8. There is no need for successive shift instructions, since up to 15 shifts can be performed in a single instruction. When the instruction is assembled, bits 0-3 contain the number of shifts to be performed. None of the shift instructions affect the Link.

The number of shifts can be taken from the vertical display scale switch instead of from bits 0 - 3 if bit 15 is set in a shift instruction. The bit assignments controlling the various shift instructions are shown below:



F. Test Instructions

The 1080 can test for several conditions and generate program branches when these conditions exist or do not exist. The test for zero is part of the Group I instructions and generates a skip of the next instruction if the calculated quantity is zero:

```
A-MZ TEST1     /SUBTRACT TEST1 FROM AC AND SKIP IF RESULT IS ZERO
JMP A          /JUMP TO A IF NON-ZERO
JMP B          /JUMP TO B IF ZERO
```

It is possible to execute (EXCT) or skip (SKIP) on each of the following conditions:

ZAC	Zero accumulator
MOAC	Minus one accumulator
POAC	Plus one accumulator
ACØ	Bit Ø of the AC = 1, useful to test for odd or even numbers, or rotation overflow
AC19	The sign bit = 1, the number is negative
L	The Link is one

The program can thus perform a skip when any of the above conditions is either true or false. For instance, SKIP AC19 means skip if sign bit is one, while EXCT AC19 means do not skip if AC bit 19 is one, but do skip if AC19 is zero. In other words the next instruction is executed (EXCT) if and only if the condition AC19 = 1 is met. This is best illustrated by the example below. The program accepts a character from the keyboard and allows it if and only if that character is an octal number. If it is an octal number, the program exits from the subroutine with that number in the AC. Note that the ASCII code for integers is biased by 260_8. The subroutine TYPE (page 32) is not shown again here.

```
OCTIN,  Ø
        JMS ECHO        /GET AND ECHO CHARACTER FROM TELETYPE
        A-MA (260       /SUBTRACT ASCII BIAS
        EXCT AC19       /IS THE RESULT LESS THAN ZERO?
        JMP ERR         /YES, TYPE AN ERROR MESSAGE
        A-MA (10        /IS THE RESULT GREATER THAN 7?
        SKIP AC19       /NO, LEGAL OCTAL NUMBER
        JMP ERR         /YES, ILLEGAL INPUT
        A+MA (10        /RESTORE NUMBER BY ADDING 10
        JMP @ OCTIN     /AND EXIT FROM THE SUBROUTINE
ERR,    MEMA (277       /ASCII FOR QUESTION MARK
        JMS TYPE        /TYPE QUESTION MARK
        STOP            /AND HALT
ECHO,   Ø               /GENERAL PURPOSE TELETYPE INPUT ROUTINE
T1,     TTYRF           /WAIT FOR READER
        JMP T1
        RDTTY           /GET CHARACTER
        JMS TYPE        /AND TYPE IT
        JMP @ ECHO
```

This program gets one character from the keyboard, and prints it using a TYPE routine such as the one in Section D. It then examines it to see if it is in the right range: $\emptyset \leq n \leq 7$. If the character is less than Ø the ASCII typed will be less than 26Ø, so that when the ASCII bias is removed, the result will be negative. This produces a jump to the ERR routine where a question mark is typed. If the result is positive (remember zero is positive), the number is tested for being greater than 7. If 10_8 is subtracted from any legal number, the result should be negative. In this case, the number is accepted. If the result is positive (or zero) the error routine is executed.

G. Miscellaneous Instructions

The miscellaneous group contains the instructions that operate on the Link. As mentioned earlier, the Link is a one-bit register which operates as an extension to the accumulator, so that when overflow or carryout occurs, the state of the Link changes. The Link is changed by an addition only if the result is transferred back to the AC. Thus A+MM does not change the Link, but A+MA does. Since this change is only meaningful if the original state is known, the following instructions can be used on the Link:

CLL	Clear the Link: set it = Ø
STL	Set the Link = 1
TLAC	Transfer the Link to AC bit 19. The Link and bits 0 - 18 of the AC are unchanged
TACL	Transfer bit 19 of the AC to the Link. Bit 19 is unchanged

It is also important to recognize that ZERA and MTOA change the state of the Link.

Finally, the instruction STOP halts the stored program processor at the end of a memory cycle.

A simple routine to add two numbers and test for overflow would be the following:

```
CLL            /CLEAR THE LINK
MEMA NUM1      /GET THE FIRST NUMBER
A+MA NUM2      /ADD THE SECOND NUMBER TO IT
SKIP L         /IS THE LINK = 1?
JMP NOFLOW     /NO OVERFLOW FOUND
JMP OVRFLW     /YES, OVERFLOW FOUND
```

H. Exercises

1. Examine the program below and decide what observable task it performs.

```
START,   ONEA
CØ,      ACCM SAVE
         MEMA K
         ANGM COUNT
         MEMA SAVE
C1,      MPOMZ COUNT
         JMP C1
         LLSH 1
         JMP CØ
COUNT,   Ø
K,       4ØØØØ
SAVE,    Ø
```

2. What will the contents of the AC be when this program halts?

```
*Ø
START,   MEMA (6
         A+MA (7
         JMS DUMMY
         LLSH 1
         A-MA DUMMY
         STOP
DUMMY,   Ø
         JMP @ DUMMY
```

3. What will the contents of TEMP be when, if ever, this program halts?

```
START,   MEMA (6
         A-MAMZ TEMP
         JMP START
         MCPM TEMP
         STOP
TEMP,    Ø
```

4. Write a program to type out THIS PROGRAM WORKS! on the Teletype. Arrange it so that each 20-bit data word contains two ASCII characters.

5. Write a program to add and subtract alternate numbers starting at some point in memory, which will halt only if the AC becomes zero. In other words, add the first location, subtract the second, add the third to the AC and so forth.

6. Write a program to punch out an endless string of "paper dolls" on the Teletype punch. Use the design below or design your own.

```
    ooo      ooo
   o   o    o   o
    ooo      ooo
     o        o
 oooooooooooooooooo
oooooooooooooooooooooooooooooooo
     o        o
    o o      o o
    o o      o o
```

I. Hardware Multiply-Divide

The Hardware Multiply-Divide logic utilizes an additional register, called the Multiplier-Quotient Register or MQ. It is used to extend the accumulator to contain double precision integers. The instructions are described below:

TACMQ	Transfer the AC to the MQ, AC unaffected
TMQAC	Transfer the MQ to the AC, MQ unaffected
BITINV	Bit Invert the AC (used in Fourier transform routines) Bit inversion means that bit 19 is interchanged with bit 0, bit 18 with bit 1 and so forth. This can sometimes be used to take reciprocals.
ZRAM	Zero the AC and MQ
MULT	The 20-bit number contained in the MQ is multiplied by the number contained in the location following the MULT instruction. The state of the AC is unimportant. At the completion of the instruction, the result is contained in the AC and MQ, with the high order part in the AC and the low order 20 bits in the MQ.

To multiply 3 by 4 the following code would be used:

```
         MEMA (4            /GET 4
         ACCM MPLCND        /STORE IN LOCN FOLLOWING MULT
         MEMA (3            /GET 3
         TACMQ              /PLACE IN MQ
         MULT               /PERFORM MULTIPLICATION
MPLCND,  Ø                  /LOCATION OF MULTIPLICAND
         ACCM HIWORD        /HIGH WORD IN AC; STORE IT
         TMQAC              /GET LOW WORD
         ACCM LOWORD        /AND STORE IT
```

In the above program, the value 4 is loaded into the AC using the immediate mode MEMA (4. The 4 is then placed in the location following the MULT instruction. This location has the label MPLCND. The value 3 is then loaded into the AC and transferred to the MQ. The MULT instruction then multiplies the contents of the MQ (3) by the contents of the location following the MULT (4). The result, which may be 40 bits long, is contained in the AC and MQ. The AC contains the high order part, in this case Ø, and it is stored in HIWORD. The MQ is transferred to the AC and the result, in this case 14_8, is stored in LOWORD.

The following two instructions are used by the division logic:

RISH n — Right Integer shift. Right shift of AC, with least significant bits dropped off right end. ($Ø \leq$ shifts $\leq 17_8$)

DIVD — Integer divide. The 38 bit dividend placed in the AC and MQ <u>left shifted one place</u>, is divided by the contents of the location following the instruction. At the conclusion of the operation, the quotient is in the MQ and the remainder in the AC. The remainder is left shifted one bit, the quotient is correct as it appears.

The reason for the shifting instructions is that it makes the treatment of numbers in the floating point format more efficient. This is utilized in the Floating Point Package version N11-20823.

For single precision division, especially in cases where the remainder is unimportant, the simple code below is representative:

```
/ROUTINE TO DIVIDE SINGLE PREC # DIVDND BY DIVSOR

     MEMA DIVSOR        /GET THE DIVISOR
     ACCM D1            /PUT IT IN DIVD LOCATION + 1
     MEMA DIVDND        /GET THE DIVIDEND
     LASH 1             /AND LEFT SHIFT IT
     TACMQ              /LOAD MQ
     ZERA               /CLEAR AC
     DIVD               /DIVIDE BY D1
D1,  Ø
     TMQAC              /GET THE QUOTIENT, IGNORE REMAINDER
     ACCM QUOT          /AND STORE IT
```

In the preceding case the contents of location DIVSOR is the divisor; it is loaded into the AC, and then stored in location D1, the location following the DIVD instruction. The dividend is loaded into the AC from location DIVDND and then left shifted one bit as required. This shifted result, which must still be less than 20 bits and unsigned, is then stored in the MQ. Most important, the AC is now zeroed. If it were not, the dividend would be the double precision AC-MQ, where the AC is merely the duplicate of the MQ. The division of the contents of the MQ by the contents of D1 is then performed with the resulting quotient in the MQ and the remainder in the AC. The remainder is ignored and the quotient is transferred from the MQ to the AC and then stored in location QUOT.

In the case of double precision division, it is necessary to shift both words left one place. This is most easily done by checking bit 19 of the low order part before the left shift. If bit 19 is one, then this one must be transferred to bit Ø of the high-order word. In the example below the link is used as a flag to indicate whether bit 19 of the low order word was set or not. The link is cleared and then bit 19 of the low word tested. If it is one, the link is set. Then the shift is performed and the shifted word transferred to the MQ. The high-order word is loaded into the AC and shifted left one place. Then, if the link is set the shifted AC is incremented by one. This sets bit Ø to one if bit 19 of the low order word was one. In the example, it is assumed that location DIVSOR already contains the divisor.

The division is then performed and the remainder appears in the AC. The remainder is right shifted one place and stored in location REMNDR. Note that since this entire division process is unsigned, the right shift is integer rather than arithmetic. Finally, the MQ is retrieved and stored as the quotient in QUOT. This is illustrated below:

```
/ROUTINE TO DIVIDE THE DOUBLE PRECISION NUMBER HIDIV - LODIV
/BY DIVSOR

        CLL                 /CLEAR LINK
        MEMA LODIV          /GET LOW ORDER WORD
        EXCT AC19           /IS BIT 19 SET?
        STL                 /YES, SET LINK AS FLAG
        LASH 1              /SHIFT LEFT ONE PLACE
        TACMQ               /PLACE IN MQ
        MEMA HIDIV          /GET HIGH-ORDER WORD
        LASH 1              /AND SHIFT IT
        EXCT L              /TEST LINK
        APOA                /INCREMENT AC IF BIT 19 WAS SET IN DBLDVL
        DIVD                /EXECUTE THE DIVISION
        DIVSOR, nnnnnn      /NUMBER ALREADY STORED HERE FOR DIVISOR
        RISH 1
        ACCM REMNDR         /SHIFT AND SAVE THE REMAINDER
        TMQAC               /GET THE QUOTIENT
        ACCM QUOT           /AND SAVE IT
```

It should be noted that the hardware multiplication and division is unsigned, and that the signs must be tested for independently by the user.

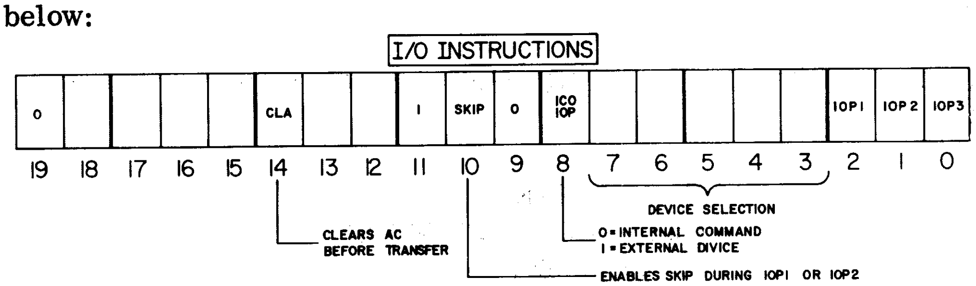
An inclusive OR operation is possible between the AC and MQ. The octal code 4341 (which has not been assigned a mnemonic) performs this logical OR operation. As with every other 1080 instruction, the source (MQ) is unchanged, and only the destination (AC) is changed.

For instance, to perform an inclusive OR between 2136412 and 0176310 the following code is used:

```
          MEMA OR1        /GET FIRST NUMBER
          TACMQ           /LOAD MQ
          MEMA OR2        /GET SECOND NUMBER
          4341            /PERFORM OR OPERATION
          ACCM RESULT     /RESULT OF OR IN AC, STORE IT
          STOP
OR1,      2136412
OR2,      0176310
RESULT,   Ø
```

J. General Input-Output Instruction Format

The input-output or I/O instructions of the 1080 utilize the bit assignments given below:



Bits 3 - 7 specify a particular device code, and bits 2, 1 and Ø specify one of three pulses called IOP1, IOP2 and IOP3. Each device can therefore have up to three signals sent to it, associated with these three IOP's. Furthermore, one computer instruction can issue all three of these pulses, since the occurrence of these signals is dependent only on whether bits Ø, 1 and 2 are set.

Bit 14 has a special function in I/O transfers. If it is set, the AC is cleared before the transfer. If it is not, a logical inclusive OR between the AC and the device buffer occurs during input. Thus, RDTTY = 44453, so that the AC is cleared before the loading of the AC from the Teletype occurs.

K. Hardware Access Instructions

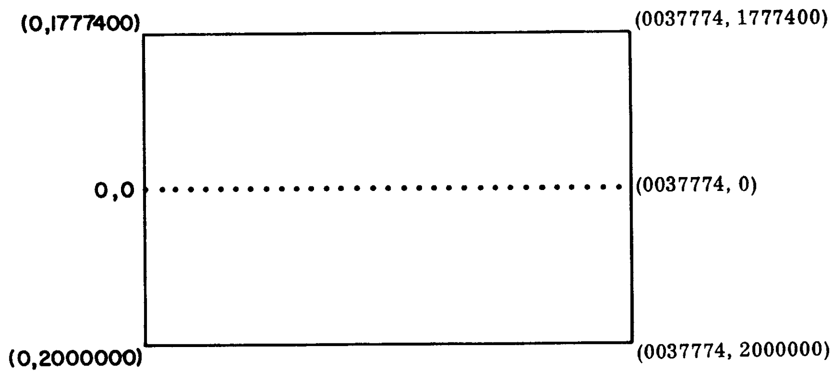
1. Display Instructions

The display instructions center around two digital to analog converters for the x and y axes of the oscilloscope. These are devices that can be loaded from the computer with digital information which is then converted to an output voltage to drive the scope. Both converters are 12-bits, with 14 bits available as an option. In both the x and y axis, the additional two bits, if added, affect the two least significant bits of the register.

The x-axis is controlled by the following instructions:

Octal Code	Mnemonic	
214001	TACXD	Transfer the AC to the x-display register. Bits 2 - 13 of the AC are transferred to the x-axis digital to analog converter. The AC is complemented during the transfer.
4014	INCXD	The x-axis is incremented by one each time this instruction is issued. This instruction is affected by the Horizontal Display scale switch, so that 1024 increments will be full scale in the 1K position, 2048 in the 2K and so forth.
4012	TACYD	Transfer the AC to the y-display register. Bits 8 - 19 of the AC are transferred to the y-axis digital to analog converter. This axis is signed: midscale is zero.
4011	INTENS	Intensify point. This instruction issues a pulse to the z-axis connector (pin J2) at the rear of the 1080 which will allow z-axis modulation of the display.
105000	VDSH	Vertical Display scale shift. The data in the AC is shifted left (arithmetic) one place for each position the Vertical Display Scale knob is set back from the 131K position.

The coordinates of the scope display in terms of the numbers that must be set into the AC are given below for the 12 bit converters.



X-Y COORDINATES OF DISPLAY
USING 12-BIT DAC'S

Programming the display is extremely simple, since it is not generally necessary to consider either which bits are in which positions in the y-axis register nor what the value of the x-axis display is. The y-axis vertical display scale is simply controlled from the Vertical Display Scale switch using the command VDSH. The x-axis is set to -1 at the beginning and need only be incremented up to the end of the loop. The following code allows the display of the first 2K of data memory. Note in particular that since INCXD, TACYD and INTENS share the same I/O device code 01, the three instructions can be issued <u>at once</u>. The x-axis is incremented before intensification, so that it is set to -1 at the beginning of the loop.

```
/SOFTWARE CONTROLLED DISPLAY PROGRAM

          *Ø
START,    MEMA PNTSET        /SET DATA POINTER
          ACCM POINT
          MEMA CNTSET        /SET COUNTER TO 2048 POINTS
          ACCM COUNT
          MONA               /SET X-AXIS TO -1
          TACXD
LOOP,     MEMA @ POINT       /GET FIRST DATA POINT
          VDSH               /SHIFT ACCORDING TO VERTICAL DISPLAY
                             /SCALE KNOB
          TACYD INCXD INTENS /LOAD Y, INCREMENT X, & INTENSIFY
          MPOM POINT         /INCREMENT POINTER
          MMOMZ COUNT        /DECREMENT COUNT, TEST FOR DONE
          JMP LOOP           /DO NEXT POINT
          JMP START          /RESET POINTERS AND START AGAIN
PNTSET,   100000
POINT,    Ø
CNTSET,   4000
COUNT,    Ø
```

2. Digitizer Instructions

There are three instructions associated with the digitizer plug-in. These reset the digitizer, start the digitizer and read it into the AC. Since all three of these share the same device code they can be combined so that the digitizer can be read, reset and started in one instruction.

Octal Code	Mnemonic	
4371	REDS	Reset digitizer
4372	STDG	Start digitizer
44374	RDG	Read digitizer into AC

One analog to digital conversion takes 20 microseconds. It is therefore necessary that there be a delay of at least 20 microseconds between the issuing, the STDG command and the RDG command. Since one instruction takes 4 microseconds (and indirect addressing takes 6) this is not difficult to arrange. It should be noted that the combination instruction RDG REDS STDG (octal code 44377) reads the last conversion into the AC and starts a new one. The digitizer input is signed: zero volts corresponds to Ø, + full scale to 377 (in the 9-bit position) and full scale negative to 3777400. The AC is filled with ones from bit 19 if the input is negative.

3. Sweep Ramp and Clock

There is a third digital to analog converter associated with the sweep plug-in, called the Sweep Ramp. It is a 12 bit DAC which can be zeroed and incremented only. During hardware data acquisition it is reset and incremented synchronously with each sweep. The two instructions are

RSWP	reset sweep ramp (to Ø)
ASRMP	advance (increment) sweep ramp

It is also possible to detect the dwell time clock flag under software control. In order to detect the clock, it must be enabled by a RSWP (reset sweep ramp instruction). It will then be started either by setting the trigger switch to auto-recur or by triggering the sweep from the trigger input pins. The instruction

DWSK	skip on dwell

allows the programmer to test the dwell clock flag. This flag stays high for 20 microseconds, so the wait loop must be no longer than two instructions to insure that the flag will be detected. Once the clock is started it runs continuously.

4. Software Control of Measure Mode

It is possible to have the stored program processor initiate the wired measure program using the command 4306. This I/O command is not defined in the Assembler but can be defined in the program using the equals sign:

```
SETM = 4306          /SET MEASURE
```

A flag is set when this I/O command is executed so that when the STOP command is given, the Measure mode will automatically commence. When the wired processor has completed the number of sweeps set on the Autostop switch, the sweep counter will be reset to zero and control returned to the stored program at the location following the STOP. A typical program, then, is:

```
SETM=4306        /DEFINE SYMBOL AS IO COMMAND 4306
     .
     .
     .           /PROGRAM TEXT
SETM             /SET MEASURE
STOP             /STOP WIRED PROCESSOR AND BEGIN MEASURE
                 /PROGRAM
MEMA TEMP        /STORED PROGRAM CONTINUES HERE AFTER N
                 /SWEEPS
```

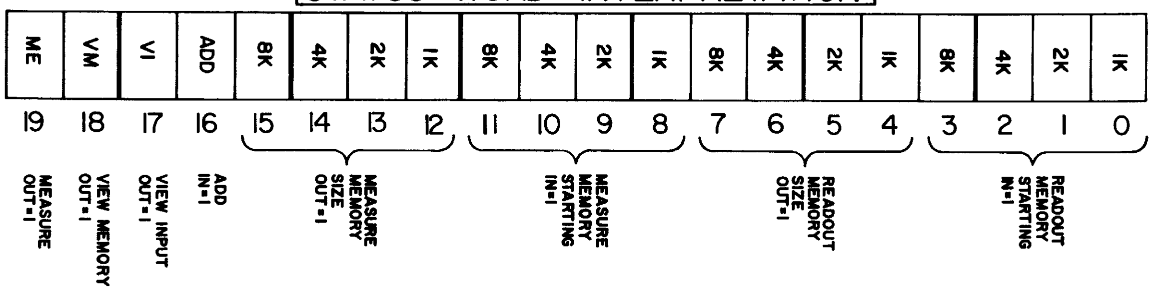
It is also possible to initiate the wired program and then halt, by giving the I/O command 4302, which simply initiates the measure program and then causes the processor to stop when N sweeps have been completed.

This feature is standard on SD-82 plug-ins with serial numbers higher than 52, on all SD-81 plug-ins and all NIC-80's. Earlier models require a minor modification. Please contact the factory for details.

5. The STATUS Instruction

The instruction STATUS causes the contents of the status register to be read into the AC. This register contains the information on which of the push-buttons specifying readout and measure starting and size are depressed. Two things should be observed about this register. It does not specify whether the 16K buttons are depressed. If none of the other size buttons are depressed, then it is to be assumed that the 16K button is depressed. The user's program must trap for this condition. Secondly, it should be noted that the Starting buttons register one if they are in, while the size buttons register one if they are out. As a result the status word must be complemented before the Size buttons are interrogated.

STATUS WORD INTERPRETATION



A typical program for examining the readout Starting and Size pushbuttons and then displaying the result is shown below:

```
/SOFTWARE DISPLAY FROM PUSHBUTTONS

START,      STATUS              /READ STATUS WORD
            LASH 12             /SHIFT OVER READOUT STARTING BITS
            ANDA MASK           /MASK THEM OUT
            A+MA DSTART         /ADD ON 100000
            ACCM POINT          /STORE ADDRESS OF FIRST DATA POINT
            STATUS
            LLSH 6              /SHIFT OVER SIZE BITS
            ACPA                /COMPLEMENT RESULT
            ANDA MASK           /MASK THEM OUT
            EXCT ZAC            /IF Ø, SET TO 16K
            MEMA K16K           /IF READOUT BITS = Ø, DISPLAY 16K OF DATA
            ACCM COUNT
DISPLA,     MONA
            TACXD               /SET X-DISPLAY REGISTER TO -1
LOOP,       MEMA @ POINT        /GET EACH DATA POINT
            VDSH                /SHIFT FROM VERTICAL DISPLAY KNOB
            TACYD INCXD INTENS
            MPOM POINT          /INCREMENT POINTER
            MMOMZ COUNT         /DECREMENT POINTER
            JMP LOOP            /DO MORE
            JMP START           /DONE, REREAD SWITCHES
MASK,       36000               /MASKS OUT ALL BUT BITS 10-13
DSTART,     100000              /ADDRESS OF FIRST DATA MEMORY POINT
POINT,      Ø                   /DATA POINTER
COUNT,      Ø                   /NUMBER OF POINTS TO BE DISPLAYED
K16K,       40000
```

L. Exercises

(1) Write a program to display 4K of data memory under software control. The keyboard should be active during the display, echoing all typed characters. When a Return is typed, the Teletype should echo with a CRLF, and when a $ sign is typed, the program should halt. Flow chart the program carefully.

(2) Write a program to solve y = mx+b for m, x and b stored in memory. Assume that y will only be single precision. Halt with y in AC.

(3) Write a program to solve y = ab/c for a, b and c stored in memory. Halt with low order part of y in the AC.

(4) Write a program to accept two positive decimal numbers from the Teletype and add them. Use hardware multiply-divide instructions.

(5) Write a program to display a horizontal line on the scope whose position depends on the input to the digitizer.

(6) Write a program to count the number of rotations of the Vertical Display Scale switch from the highest position, and halt with the number in the AC.

(7) Explain how the instruction ZERZ is used below:

```
START,  JMS READ
        A-MZ (301
        ZERZ
        JMP A
        A-MZ (302
        ZERZ
        JMP B
        JMP START
    A,  --- ---
        .
        .
        .
    B,  --- ---
        .
        .
        .
```

TABLE I

GROUP I INSTRUCTIONS

Octal	Mnemonic	Source (Operator)
0500000	A+M	Add accumulator (AC) and memory
0520000	AMP	Add accumulator, memory and 1
0460000	A-M	Subtract memory from the accumulator
0320000	M-A	Subtract the AC from memory
0440000	ACM	Accumulator plus the complement of memory
0300000	CAM	Complement of the AC plus memory
0000000	AND	Logical AND between accumulator and memory
0100000	MEM	Take the contents of memory
0120000	MPO	Memory plus 1
0700000	MMO	Memory minus 1
0040000	MCP	Complement of memory
0060000	MNG	Negative of memory
0400000	ACC	Accumulator
0420000	APO	Accumulator plus 1
0540000	AMO	Accumulator minus 1
0200000	ACP	Complement of the accumulator
0220000	ANG	Negative of the accumulator
0160000	ZER	Take the number zero*
0020000	ONE	Take the number 1
0140000	MON	Take the number -1
0740000	MTO	Take the number -2*
0000000	JMP	Jump
2000000	JMS	Jump to a subroutine, leave PC in first address

*These instructions, if loaded into the AC, will change the state of the Link.

Destination (Suffix)

0010000	A	Accumulator
0004000	M	Memory
0002000	Z	Zero test unit Skip if result is 0

Addressing

0000000	=	immediate
2000000	=	direct
3000000	=	indirect

GROUP II INSTRUCTIONS

Shift Group

0005000	LASH n	Left arithmetic shift of AC, Link unaffected
0005020	RASH n	Right arithmetic shift
0005040	LLSH n	Left logical shift
0005060	RLSH n	Right logical shift

$0 \leq n \leq 17_8$

Skip Group

0005100	SKIP on	0400020	ZAC	Zero AC
0005140	EXCT on	0420000	MOAC	Minus one AC
		0540020	POAC	Plus one AC
		0000010	ACØ	AC bit Ø = 1
		0000004	AC19	AC bit 19 = 1
		0000001	L	Link = 1

Miscellaneous Group

0005220	STOP	Processor halts
0005210	CLL	Clear the Link
0005204	STL	Set the Link = 1
0005202	TLAC	Transfer the Link to bit 19 of the AC, bits 0-18 and Link unchanged
0005201	TACL	Transfer bit 19 of the AC to the Link, bit 19 unchanged

Input-Output Instructions

0006454	TTYRF	Skip when the Teletype keyboard reader is ready
0044453	RDTTY	Read the Teletype keyboard-reader buffer into the AC
0006444	TTYPF	Skip when the Teletype printer is ready for a new character
0004443	PRTTY	Print the character contained in the AC
0006464	HSRF	Skip if the high speed reader is ready
0044463	RHSR	Read the next character from the high speed reader into the AC
0006474	HSPF	Skip if the high speed punch is ready
0004473	RHSP	Punch the contents of the AC

Hardware Access Instructions

0105000	VDSH	Shift AC by amount controlled by vertical display scale switch
0004371	REDS	Reset Data Scaler to start new digitization
0004372	STDG	Start digitization
0044374	RDG	Read Digitizer result into AC
0006362	DWSK	Skip on Dwell Time flag
0004361	ASRMP	Advance Sweep Ramp
0004364	RSWP	Reset Sweep Ramp
0044034	STATUS	Read hardware settings into AC.

Automatic Arithmetic Instructions

0505320	MULT	Multiply contents of MQ by next location
0465300	DIVD	Divide AC-MQ by contents of next location
0004354	TACMQ	Transfer AC to MQ
0004343	TMQAC	Transfer MQ to AC
0044354	ZRAM	Zero AC and MQ
0004347	BITINV	Bit invert the AC
0405000	RISH	Right integer shift of AC

Display Instructions

0214001	TACXD	Transfer AC to X display register (AC is complemented)
4012	TACYD	Transfer AC to Y display register
4014	INCXD	Increment register
4011	INTENS	Intensify display

IV. LOADING PROGRAMS INTO THE 1080

A. Pushbuttons

On the bottom of the section of the 1080 labeled "290 Display Control" there are seven rectangular pushbuttons which stay in when pressed and two square buttons, marked Execute and Stop, which do not. When one depresses one of the seven buttons he indicates which function he wishes to perform. Pressing Execute actually causes this function to be performed.

Directly above the pushbuttons are 20 toggle switches, called the Switch Register. Twenty bit binary numbers can be represented in the Switch Register, where the up position represents a one and down a zero. These switches can be used to specify memory addresses and data to be deposited in memory. The use of these switches is discussed in detail below. They are shown in the photograph on page 15.

LOAD PC -- Depressing this button, followed by pressing Execute causes the contents of the Switch Register to be transferred to the Program Counter (or PC). This value is also loaded into the AC at the same time, although this is of little general use.

CONTINUE -- Pressing Execute when this button is depressed causes the Stored Program processor to begin interpreting instructions at the address specified in the Program Counter. Thus, using the combination LOAD PC, Execute, CONTINUE, Execute, the processor can be started at any address.

START -- This button causes the processor to begin executing instructions at location Ø. It is equivalent to loading the PC with ØØØØØØØ and then pressing CONTINUE followed by Execute. One can also start programs at location zero using the Stored Program Start pushbutton on the 1080 console.

SINGLE INS -- If the processor is running, depression of this button will cause it to stop at the end of whatever instruction it is performing. Then, each time Execute is depressed the processor will execute one instruction. One can execute single instructions from any arbitrary address by loading the PC with that address, depressing SINGLE INS, and then pressing Execute once for each instruction to be executed.

EXAMINE -- The contents of the location whose address is in the PC are loaded into the AC for examination. If a number of sequential locations are to be examined, the red button STEP should be depressed. STEP causes the PC to be incremented automatically, so that each time Execute is depressed the next sequential location is displayed in the AC.

DEPOSIT -- The contents of the Switch Register are loaded into the memory address specified in the PC. Thus, to deposit a number in memory, one sets the address into the switch register and depresses LOAD PC followed by Execute and then

(5) Checksum - At the end of each block of sequential data, the checksum is punched. It is the lowest order 20 bits of the running sum kept of that data block. It differs from actual load data only in that it has column 7 punched as well as columns 0 - 6. Following the checksum may be either a new load address or trailer code.

(6) Trailer Code - This is identical to leader tape, except that it may have a Rubout punched in it. A rubout punched in pure trailer tape is a signal for the Binary Loader to halt.

Leader	10000'000
Leader	10000.000
.	.
Load Adress 1	0
Load Adress 2	0
Load Adress 3	0
Data Word 1	0
Data Word 2	0
Data Word 3	0
.	
Checksum 1	1
Checksum 2	1
Checksum 3	1
Trailer	10000.000
Trailer	10000.000
u.s.w.	
Rubout	11111.111
Trailer	10000.000
u.s.w.	10000.000

V. THE ASSEMBLER-EDITOR

A. Introduction

The Nicolet Assembler-Editor, 1973 (NIC-80/S-7304) is a program which translates mnemonic codes into binary information in a form suitable for read-in by the Binary Loader. It also produces a listing of each address, its octal contents and corresponding octal code, and any comments.

In addition to these capabilities, this program is also a text editor. The ASCII code representing the mnemonics is stored in data memory and can be altered and corrected there.

B. Preparation of Source Tapes

There are two ways to prepare source programs for the Assembler. The first method is to punch the tape out using the Teletype in the LOCAL mode, while the computer is performing some other task, such as signal averaging or data reduction. While the Teletype is in the LOCAL mode, the tape can be punched without affecting any concurrent computer operation. If an error is made while typing, backspace the tape as many times as there are illegal characters, and then type a RUBOUT for each character to be delted. This procedure overpunches a rubout (octal 377) on each tape line. When the tape is complete it can be read in by the Assembler. During read-in mode, rubouts are ignored by the Assembler-Editor program.

The second method of preparing a source tape is using the Editor itself. In the Insert mode, it is possible to create new programs by inserting at line 1 over a previously created "empty" program. When the program is completed, exiting to the Assembler will allow an immediate error analysis of the program.

C. Logic of the Assembler-Editor

The Assembler is a fairly simple program which examines each line of text stored in memory and decides how to translate it to the octal equivalent. If the instruction is a Group II instruction, the translation procedure is to consider each word separately. For instance if the instruction SKIP AC19 is encountered, the octal code for SKIP is found to be 5100 and the code for AC19 is found to be 0004. The two values are ORed together and the result punched on paper tape or listed on the Teletype.

If the instruction is found to be a Group I instruction, such as MEMA TEMP, the code for MEM (0100000), the code for direct addressing (2000000), and the code for suffix A (0010000) are combined to produce the code 2110000 for MEMA. Then the text is scanned for a definition of the address TEMP. The Assembler must find a line containing TEMP followed by a comma. As it is searching for this line, it keeps track of the address of each line and when it finds TEMP it takes the right hand ten bits of that address and ORs them with the calculated value for MEMA. If it found that TEMP was defined at address 5365, for instance, it would combine 2110000 with 1365 to get 2111365 for MEMA TEMP. This value is then either punched out in binary form or listed on the Teletype.

The Editor stores characters in the first 8K of data memory, three characters per word, in horizontal order. Each ASCII character is converted to packed six bit form by subtracting 240 from it. If the result is less than zero, the result is ignored. If the result is greater than zero, the six bit result is saved and stored in a data word. Characters are stored in bits 17-12, 11-6 and 5-Ø of each successive word. A new line is flagged by starting a new word and putting a 77 in bits 17-12, representing a Return. The last character of any text must be a dollar sign. It cannot appear at any other place in the listing. When the Assembler or Editor encounters a dollar sign, it assumes that is the end of the text and proceeds no further.

Since the code for a carriage return is 215 and therefore less than 240, the Return is represented by the code 77. The back arrow character cannot be used in these texts since 337 - 240 also = 77_8.

D. Assembler Conventions

1. Special Characters

The following special characters are recognized by the Assembler:

/	Starts a comment. All characters beyond the slash except $ are ignored by the Assembler until a carriage return is encountered.
$	Signifies the end of the program. Cannot appear elsewhere in a tag, instruction or comment.
,	Designates a tagged address; the first six characters following the previous Return and before the comma are taken as the name of the tag. A tag need not be six characters, but any after six are ignored.
(	Designates immediate address mode.
space	Used to separate operators from operands. There must not be a space between the operator and its suffix.
@	Designates an indirect instruction. An indirect immediate instruction is flagged as an error.

* Designates the starting address of the code that follows.

= Allows the definition of symbols. For example, TTY2RF = 6434 defines the flag of a second Teletype, with I/O code 43.

The Assembler recognizes only printing characters as meaningful. All non-printing characters are ignored.

2. Syntax

a. Spaces may be used freely to improve legibility. They are not required anywhere except between a Group I instruction and its operand. Their omission here will generate the error message IS (Illegal Suffix).

b. A comment may contain any character except a dollar sign.

c. All non-printing characters are ignored on input and are not stored.

d. Numbers can be entered only as positive octal integers.

e. Labels

i. Must be separated from the location contents by a comma. No space is required following the comma, but is recommended for legibility reasons.

ii. Must start with an alphabetic character but may contain any combination of alphabetic and numeric characters after the first.

iii. May contain up to six characters. Labels differing only in characters beyond the sixth are treated as identical.

iv. May <u>not</u> contain embedded spaces.

E. <u>Assembler Loading and Use</u>

1. <u>Loading</u>

The Assembler-Editor tape (NIC-80/S-7034) is loaded using the standard Binary Loader. The program is started at 2000 as follows:

a. Set the Switch Register to 2000 (00 000 000 010 000 000 000).

b. Depress LOAD PC and press Execute.

c. Depress Continue and press Execute.

The program will start by typing a carriage return, line feed, and the program name ASSEMBLER. It is then ready for commands.

2. Assembler Commands

R - Read in a source tape. If the source tape has been prepared off-line or if a tape has been generated previously by the Editor, the command will cause tape to be read in from the low speed reader unless there is tape in the high speed reader. In this case the high speed reader is automatically used. The tape does not echo during read-in, and all non-printing characters, such as Rubout, are ignored. The tape must end with a dollar sign in order to terminate the read-in routine.

E - Perform an error analysis. This command causes the Assembler to try to assemble the entire text stored in memory without printing or punching any output. If the text is fairly lengthy, the error analysis may take several seconds. When the analysis is complete the program will type a dollar sign.

If errors are detected, they fall into one of the following categories:

IS - Illegal Suffix
A suffix other than A, M or Z has been detected. The usual cause is no space between operator and operand.

II - Illegal Immediate
The M suffix has been used in Immediate mode, or an indirect immediate has been found.

NL - No Label
The label has not been defined.

DL - Duplicate Label
Two or more labels have been found that are identical in the first six characters.

DU - Don't Understand
All other illegal syntax and untranslatable text. This includes Group I instructions without any suffixes at all, as well as most typographical errors.

Note that there is no trap for direct addressing of constants on another page or for logical errors that are executable but meaningless.

SO - Symbol Table Overflow
More than 341 labels used.

NR - No Room
Text memory full.

B - Punch a binary tape of the stored text. The low speed punch should be turned on before giving this command. Leader and trailer are automatically punched as well. If a high speed punch is available, the command HB will cause the output to be on the high speed punch. A longer leader is automatically produced.

L - List the assembled code on the Teletype. The Assembler lists the address, octal contents, mnemonic and comment for each line of text. Lines which are not assembled, such as blank lines or those containing only comments are also listed although, of course, without any octal information preceeding them.

S - List out the symbol table of the current program.

H - This command, when prefixed to any of the Assembler commands, causes the resulting output to be on the high speed punch instead of on the Teletype. The command HL, for instance, produces a listing on the high speed punch.

CTRL/E - Enter the Editor. The command CTRL/E is produced by holding down the CTRL key and typing E. This command causes the Assembler to enter the Editor mode, pause while locating the end of the text, and type out EDIT.

3. Editor Commands

The Editor is line-oriented. Each line of text can be accessed as a unit having an octal number. If lines are inserted or deleted the number of each line following the change will automatically be updated.

Lines can be printed for examination, inserted before a given line, or deleted. In each case the line must be specified by number in one of two modes: Octal or Absolute. The line numbering system is considered to be in one of these modes at all times and changes only when specifically commanded.

If the command P (print) is issued, the Editor will type out the current line numbering mode by following the P with an O or an A. If the user wishes to change from one mode to the other he simply types A or O before entering the line number he wishes to print. The complete command is therefore PO nnnn, where nnnn is the line number in the Octal mode.

In the Octal mode, the line which will have address nnnn when assembled is printed. In the Absolute mode, all lines are numbered, regardless of content, and the line which is nnnnth in the list is printed. Lines which do not contain executable statements, such as blanks, comments, address definition or symbol definition lines can therefore only be accessed in the Absolute mode. A comparison between the two numbering systems is given below for a short program.

Absolute	Octal	Text
1		/EXAMPLE PROGRAM
2		*1ØØ
3		
4	100	START, MEMA TEMP
5	101	ACCM TEMP2
6		/NOW HALT THE PROCESSOR
7	102	STOP
1Ø		$

The actual commands in the Editor program are

Pm nnnn — Print line nnnn in mode m. If no number is specified, the last line number entered is used. Folow the line number with a Return.

Dm nnnn — Delete line nnnn in mode m. Follow with a Return.

Im nnnn — Insert text before line nnnn. The program remains in the Insert routine until the character CTRL/D is struck. Exit then occurs automatically. A carriage return is inserted last only if one is actually typed.

N — Print the next line in sequence following the previous one printed. This command does not increment the line counter.

CTRL/A — Append more text to that already stored in memory.

W — Write out the text stored in memory. If H was typed before entering the Editor, this will be done on the high speed punch. A leader and trailer are automatically punched in either case.

CTRL/L — Exit from the Editor to the Assembler.

The Editor remembers the last line number entered so that one need not retype it while operating on the same line. Let us suppose that the line

MEMQ TEMP

has been typed by accident. It is absolute line 37 and the mode is currently octal. The following series of commands prints this line, deletes it and inserts a new one.

PO A37	print absolute line 37, changing the mode to absolute
MEMQ TEMP	the line is printed on the Teletype
DA	line 37 is deleted
IA	a new line is inserted
MEMA TEMP	
(CTRL/D)	exit from the Insert routine

F. Special Features of the Assembler

The Assembler can be restarted at any time by pressing STOP and restarting at location 2000.

Since neither Read nor Append echo at the keyboard, programs can best be created from scratch by typing R, Return, Return, $ to zero previous text and create a two-line blank program. One can then insert all the text needed by starting with IA 1. The command IA Ø is not legal.

The Assembler recognizes the instruction

MEMA (LABEL

as an instruction to get the relative address of the labeled location. The operand becomes the 10 least significant bits of the address LABEL. While this has limited general use, it becomes extremely useful on page zero (locations 0-1777), where the relative and absolute addresses are identical.

TABLE II

Nicolet Assembler Command Summary

	SA = 2000
R -	Read in Source Tape
B -	Punch Binary Tape
E -	Error Analysis
L -	List Assembled Code
H -	Causes output of B, L, W to be on HSP
S -	Print out the Symbol Table
Control/E -	(WRU) Enter Editor

Editor

W -	Write out the Source Tape
P -	Print line A - Absolute O - Octal
I -	Insert EOT to Exit (Control/D)
D -	Delete
N -	Print next line
Control/A -	Append more text to buffer from LSR if ADD is depressed from HSR if SUBTRACT is depressed
Control/L -	(FORM) Exit to Assembler

`ASSEMBLER`	Program is started at 2000
`R`	The sequence R, Return, Return, $ enters a two line blank program
`EDIT`	CTRL/E enters the Editor
`IO A1`	Insert before Absolute line 1, mode changed from Octal

```
/EXAMPLE PROGRAM
*100

START, MEMA TEMP
  ACCM TEMP2
/NOW HALT THE PROCESSOR
  STOP
```

This text is entered

CTRL/D exits from the Insert routine

`ASSEMBLER`	CTRL/L exits from the Editor
`E`	Error analysis performed
`?NL AT     100`	
`  START, MEMA TEMP`	The labels TEMP and TEMP2 have not been defined
`?NL AT     101`	
`    ACCM TEMP2$`	
`EDIT`	CTRL/E re-enters the Editor
`PA 5`	
`  ACCM TEMP2`	Line 5, 6 and 7
`N`	are printed
`/NOW HALT THE PROCESSOR`	
`PA 7`	
`  STOP`	
`IA 10`	Insert after line 7

```
TEMP, 0
TEMP2, 0
```

The two constants are defined

CTRL/D to exit from Insert
CTRL/L to exit from the Editor

`ASSEMBLER`	
`ES`	No errors found
`B•" C(D`	B punches out a binary tape. The "garbage" is produced by
`        FW`	the Teletype attempting to type out binary characters.
`L`	The program is listed

`--------`	Page cutting guide every 66 lines
`/EXAMPLE PROGRAM`	Title printed at top of every page is the contents of the first line of the program text.

```
/EXAMPLE PROGRAM
*100

    100 2110103   START, MEMA TEMP
    101 2404104     ACCM TEMP2
/NOW HALT THE PROCESSOR
    102    5220     STOP
    103       0   TEMP, 0
    104       0   TEMP2, 0
```

Listing includes address, contents and mnemonic codes.

VI. DEBUGGING PROGRAMS

A. Introduction

Thus far we have concerned ourselves with the preparation of programs by logical design, flowcharting, coding and assembly. The major part of any programming effort, however, occurs after all these steps have been completed. This step is, of course, debugging. Once the programmer overcomes the feeling that a program which does not run the first time indicates a failure, he can program most efficiently. Virtually no program runs correctly when it is first written, and the more complex the logic, the greater the number of bugs that can creep in during the process. The programmer should recognize that he is less than half done when the program is coded, and plan accordingly.

B. Outline of a Well-Written Program

While it is impossible to write down a set of rules which cover all potential errors, it is possible to outline some general rules which cover most programming situations.

1. Initialization

Improper initialization of pointers, counters, constants and I/O facilities probably accounts for 80 to 90% of all program failures. To appreciate the magnitude of the problem, consider the following two programs for adding together ten numbers stored in locations 200-211:

```
START,  A+MA @ POINT          START,   MEMA PNTSET
        MPOM POINT                     ACCM POINT
        MMOMZ COUNT                    MEMA (12
        JMP START                      ACCM COUNT
        STOP                           ZERA
POINT,  200                   LOOP,    A+MA @ POINT
COUNT,  12                             MPOM POINT
                                       MMOMZ COUNT
                                       JMP LOOP
                                       STOP
                              PNTSET,  200
                              POINT,   0
                              COUNT,   0
```

While the two programs are designed to do the same job, the left hand program makes several unwarranted assumptions. The worst of these is assuming that the AC is zero when the program starts. The AC must be specifically set to zero for this to be true. However, even assuming that the AC is zeroed before the program is started at START, the left-hand program will only run once correctly. After the first time, the pointer POINT will contain 212 and the counter COUNT will contain Ø. These values are not reset by restarting the program, so that the second time it is run, numbers will be summed starting at location 212 and will continue until the location COUNT again reaches zero. Location COUNT will only be zero after one has been subtracted 1,048,576 times! This sort of error can therefore cause seemingly endless looping of a program.

Initialization of the Teletype flags is also necessary to ensure bug-free operation. As was mentioned earlier, the keyboard-reader buffer contains an indeterminate character when power is first applied to the Teletype and the flag may be in either state. One of the first commands should therefore be a RDTTY which will read contents of the reader buffer into the AC (where it should be ignored) and clear the flag.

The printer should also be initialized. The flag will be in the one state whenever the printer is not printing, but the position of the carriage will be unknown to the program. Each program should therefore also begin with the typing of a carriage return-line feed combination.

If the above criteria are satisfied, the program should be "serially reusable."

2. Routines Versus Subroutines

As a general rule, any section of code that is used more than once should be coded as a subroutine. This simply minimizes the number of memory locations required. It is also often desirable to write routines in subroutine form even if they are used only once, if writing them in this manner simplifies their division into a logical unit. This division is particularly useful when debugging or rewriting a program, since such routines can be tested and moved separately.

3. Program Gullibility

One of the principle errors encountered in writing programs is that of program gullibility. This term simply implies that a particular program expects only certain kinds of data and therefore mistreats data which does not fall within that classification. For instance, a routine to accept decimal numbers from the Teletype might just subtract 260_8 from the typed character and store it as a number, without first testing to see whether the character typed lay between 260 and 271. Thus, if a typographical error were made and a Q (321) were typed instead of a one, the program would subtract 260 from 321 and arrive at a result not equal to any possible decimal digit. All routines should allow any possible form of input. They should check for range, sign and zero.

Gullibility is not limited to input routines, of course. Any routine which assumes that a number transferred to it has some abnormal constraint such as sign or size is prone to disaster unless all of these factors are independently trapped and checked.

4. Zero Effects

It is very easy to overlook the zero case in designing a piece of logic. While zero is very often a legal input value to a routine--such as perform the following loop zero times--it is likely that the programmer will forget to test for the zero case independently. This usually causes extremely long execution times or looping.

5. End Effects

The problem of end effects is closely related to both the zero case and general gullibility, but warrants a few special comments. It is important to remember that both the first and last points of a list may require special consideration, both because of storage allocation and counter-pointer resetting problems. In general, it is good practice to set all counters and pointers before entering a loop so that if some condition interrupts execution of the loop, starting the loop over again still assures that the entire list will be processed.

6. Conditional Branching

It is very important that every time a program makes a decision based on conditions such as zero-non zero, plus-minus, odd-even, or the state of the link, that the programmer carefully check and recheck the conditions under which the branching will occur. Does the program skip on the correct condition? It is very easy, for instance, to write down SKIP AC19 when careful consideration will show that EXCT AC19 is correct.

7. Comments

Of all the features of a well-written program none is more important than extensive comments, including whole paragraphs of explanation where appropriate. It is easy to brush these off by saying they "are too hard to type" or "take too long to list," but the time spent entering comments is always far less than the amount of time needed to track down a bug in an uncommented program.

There must be enough comments to help the programmer when he is writing and revising a program, and enough so that if the need for revision occurs several months later, it will be easy to find out how a particular section of code operates.

Third, there must be sufficient commenting in a program that any other person can understand its flow. This is especially important if programs are shared between researchers at different locations.

8. Human Engineering

Finally, no program is of much value if is difficult or confusing to use. The ideal program should operate in such an obvious way that any researcher in that discipline can operate it virtually without instruction. This means that it should type out messages of explanation, that commands and constants should have names associated with their function, and that it should be easy to start, restart and modify while running. The attitude that only the original programmer will ever need to use a given piece of software has caused hundreds of thousands of man hours to be wasted when a second person must discover or rediscover the operating procedure for a program.

C. Use of Nicobug II

1. Manual Debugging

Regardless of the care which is exercised, however, most programs will exhibit one or more mysterious bugs which will require testing of the program in some way. It is necessary that a case be prepared for which the exact answers are known, either from another program or from hand calculation. Then the program must be stepped through, a few instructions at a time, and the results observed.

One method of doing this is using the switch register. The Single Step button allows the execution of one instruction each time the Execute button is depressed. The program can then be started and stepped an instruction at a time. The results can be observed by watching the AC, PC and IR lights.

This method has the obvious disadvantage that if the program is at all lengthy the single stepping procedure becomes quite tedious. However, it is possible to decrease this tedium by inserting a STOP instruction near the program section where errors are suspected and allowing the program to run freely up to this point. The computer will halt at the STOP instruction and can then be single stepped from there. Unfortunately, very few programs have space available for STOPs without the replacement of existing instructions. This means that it would be necessary to remember the actual contents of the location where a STOP is inserted and then restore them before continuing. A much more efficient method of debugging is utilizing Nicobug II, a program designed to simulate the above procedure under software control.

2. Loading and Storage of Nicobug II

Nicobug II is loaded using the standard Binary Loader. It occupies locations 4632-5365, but starts at 4700. The storage layout for the first 4K during debugging might look like this:

0 - 1777	free for user programs
2000 - 4601	Assembler - Editor
4632 - 5365	Nicobug II
5366 - 6000	free
6000 - 7577	Floating Point routines, if used
7600 - 7625	Swap
7632 - 7777	Binary Loader

However, if more than one page is needed for debugging, Nicobug II can be SWAPped to the third 4K, addresses 114632 - 115365 by running SWAP. It is then started at 114700. Nicobug II automatically relocates itself and will operate correctly at this address. Swap is referred to in the support software section of the 1080 manual, and is simply a program to interchange the contents of location 0 - 7577 with those of 110000 - 117577 (the second data stack). Running it twice in succession restores the original contents of each stack. Neither Swap nor the loaders are moved and only Nicobug II will operate correctly in both locations. Swap is started at location 7600 and takes about 0.2 seconds to perform the interchange.

3. Nicobug II Commands

The following commands are used by Nicobug II, where nnnn is used to symbolize any 20-bit octal number or address:

Command	Meaning
nnnn/	Print out the contents of address nnnn and allow modification
/	Print out the contents of the last address examined and allow modification.
(line feed)	Close any location currently being examined and print out the contents of the next sequential address for modification.
nnnnG	Load the saved AC into the AC and begin executing instructions at location nnnn.
G	Begin executing instructions at location Ø.

nnnnS	Load the saved AC into the AC and execute a subroutine beginning at location nnnn. When finished return to Nicobug.
nnnnB	Insert a breakpoint at location nnnn so that execution of this instruction will cause a jump back to Nicobug where the Link and AC will be saved. Address nnn1777 is used as a pointer address.
B	Remove the current breakpoint and restore the contents of address nnn1777. (A breakpoint at location zero is not permitted.)
C	Restore the saved AC and Link and continue from the breakpoint location.
nnnnC	Continue from the breakpoint and allow the program to loop through the breakpoint nnnn times before returning to Nicobug.
A	Print out the contents of the saved AC for modification.
F	Print out current lower limit location "From" and allow modification. Close with a carriage return.
T	Print out current upper limit location "To" and allow modification. Close with a carriage return.
M	Print out current data Mask and allow modification.
nnnnD	Dump (print) all memory locations lying between From and To which are equal to nnnn after having been ANDed with the data Mask.

4. Opening and Modifying Locations

Nicobug II is first of all a program for examining and changing memory locations in an extremely simple fashion. Any memory location, including those in Nicobug itself can be examined by simply typing nnnn/. If, after examining a memory location, you wish to change it, simply type the new value immediately after the old value and close with a Return. If you also wish to examine and alter location nnnn+1, type a Line Feed instead of a Return and the new contents will be deposited and the next location opened. Only a Return or a Line Feed are legal terminators; any other character will be regarded as an error, which will not modify that memory location. If the termination character is one of the legal commands, that command will be executed instead.

5. Breakpoints

The most useful feature of Nicobug II is the breakpoint. By typing nnnnB, one replaces instruction nnnn temporarily with a jump instruction which returns control to Nicobug II. Since the instruction may well be on another page, this jump is always an indirect one through location 1777 of the current page. For this reason, location 1777 cannot be referred to by a program while a breakpoint is in effect. Once the breakpoint is removed, both the instruction and location 1777 are restored. This obviously affects the Binary Loader if debugging is carried out on page 6000, since it starts at 7777.

When a breakpoint is in effect, each time the program passes through that point, the program jumps back to Nicobug, where the contents of the AC and link at that point in the program are typed out in the format

0001234 1;2134542

where 0001234 is the address of the breakpoint, 1 is the contents of the link and 2134542 is the contents of the AC.

Any of the Nicobug commands are then available. Memory locations can be examined and changed, allowing one to modify instructions directly in octal. The saved AC and the breakpoint itself can also be changed at this time. The location of the breakpoint can be changed to further along in the program, or the breakpoint removed altogether. When all possible information has been obtained, the C command will tell Nicobug to restore the saved AC and link and continue from the last breakpoint.

The breakpoint will remain in force and if a program loop returns to that breakpoint, control will again be transferred to Nicobug. If it is desirable to examine the program only after a number of loops through the breakpoint, the command nnnnC will allow nnnn passes before control is transferred back to Nicobug.

6. Masks and Dumps

The command nnnnD causes a dump of all memory locations between From and To which are equal to nnnn after being ANDed with the Mask. This feature can be used as a straightforward memory dump, or as a sophisticated searching technique to find locations having particular values or even particular bits in common.

To illustrate this, let us first consider the simplest case, where the Mask is Ø. If F is set to 7600 and T to 7605, then the command D will cause a dump of all locations between 7600 and 7605. If, now, we change the Mask to 3777777, the command 1605D will cause a listing of only those locations between 7600 and 7605 having the value 1605.

7. Examples of the Use of Nicobug

Dump and Breakpoint examples are given for the Swap program listed below:

```
--------
/SWAPS 0-7577 WITH 110000-117577
```

```
/SWAPS 0-7577 WITH 110000-117577
*7600
  7600 2165620  START, ZERM PRGPNT  /SET PROGRAM STACK POINTER
  7601 2111621     MEMA DSTART
  7602 2405622    ACCM DPNT      /SET DATA STACK POINTER
  7603 2111623    MEMA CNTSET
  7604 2405624    ACCM COUNT    /SET COUNTER TO 7600 WORDS
  7605 3111622  LOOP, MEMA @ DPNT  /GET PROGRAM AREA WORD
  7606 2405625    ACCM TEMP
  7607 3111620    MEMA @ PRGPNT   /GET PROGRAM AREA WORD
  7610 3405622    ACCM @ DPNT  /PLACE IN DATA STACK
  7611 2111625    MEMA TEMP  /GET DATA STACK WORD
  7612 3405620    ACCM @ PRGPNT  /AND PLACE IN PROGRAM STACK
  7613 2125622    MPOM DPNT     /ADVANCE POINTER
  7614 2125620    MPOM PRGPNT
  7615 2707624    MMOMZ COUNT  /DECREMENT COUNTER
  7616    1605    JMP LOOP
  7617    5220    STOP      /STOP AFTER 7600 WORDS DONE
  7620       0  PRGPNT, 0
  7621  110000  DSTART, 110000
  7622       0  DPNT, 0
  7623    7600  CNTSET, 7600  (7577)
  7624       0  COUNT, 0
  7625       0  TEMP, 0
```

F0000000 7600	From set to 7600
T0000000 7605	To set to 7605
M3777777 0	Mask set to zero
D	Dump of all locations between 7600 and 7605
0007600/2165620	
0007601/2111621	
0007602/2405622	
0007603/2111623	
0007604/2405624	
0007605/3111622	
M0000000 3777777	Mask set to all ones
T0007605 7625	To changed to 7625
1605D	List of all locations equal to 1605
0007616/0001605	
M3777777 2001777	Mask changed to search for memory reference instructions
2001620D	Dump of all locations referring to 7620
0007600/2165620	
0007607/3111620	
0007612/3405620	
0007614/2125620	
7603B	Breakpoint set to 7603
7600G	Program started at 7600
0007603 0;0110000	Breakpoint occurs when program reaches 7603
7605B	Breakpoint moved to 7605
C	Program continues from 7603
0007605 0;0007600	Breakpoint occurs at 7605, AC = 7600
7612B	Breakpoint moved to 7612
15C	Continue through new breakpoint 15 times (octal)
0007612 0;0162000	Breakpoint occurs after 15 loops
7620/0000014	Location 7620 examined
0007621/0110000	Successive locations examined by typing Line Feeds
0007622/0110014	
0007623/0007600	
0007624/0007564	
0007625/0162000	
B	Breakpoint removed

D. Exercises

(1) If, at the end of the Nicobug II example in the preceding section, the breakpoint is not removed, and the command 5000C is given, the program seems to run wild. Explain why.

(2) Assemble, run and debug your answers to several of the previous exercises.

(3) Explain why this program shows poor programming practices:

```
START,    MEMA @ POINT
          A+MM SUM
          JMS DUMMY     /ASSUME THIS ROUTINE IS REAL AND WORKS
          MPOM POINT
          MMOMZ COUNT
          JMP START
          MEMA PNTSET
          ACCM POINT
          MEMA (15
          ACCM COUNT
          JMP START
POINT,    5000
PNTSET,   5000
COUNT,    15
```

(4) Write a program to display a box on the oscilloscope.

(5) Write a program to sample the digitizer each time a Teletype key is struck and display the level of the input signal as a horizontal line until another key is struck.

E. NMR-80, LAB-80 and BNC-12 Commands

The following commands are unique to the NMR-80, LAB-80 and BNC-12 systems and are, except for the SETM, PULSE, SENSE and PEN LIFT commands, not available on the 1080.

4002 PENLFT	Loads the Pen enable register with AC bits 0 and 1. Bit 0 controls pen up and down (0 = up, 1 = down) Bit 1 controls plotter output enable (0 = grounded, 1 = enabled)
4301 SETKNB	Causes digitizer to read parameter knobs A or B. Bit ACØ controls the choice. 0 = B, 1 = A.
4372 STDG	Start digitizer. Starts digitizer running to read either knob.
44374 RDG	Read Digitizer. Load result into AC. Ten usec must elapse between STDG and RDG.
4031 RSCNTR	Reset Sweep counter to zero.
4032 ASCNTR	Increment sweep counter
4311 LDWELL	Load dwell time register from AC. Sets time as integer in microseconds. The minimum dwell time is 0000012, or 10 microseconds. The maximum is 3777777, treated as a positive integer, or 1.048575 seconds per point.
4312 LDELAY	Load Delay register from AC. Zero is a legal delay. The maximum is the same as LDWELL.
4302 SETM	Set Measure flip-flop. Next Stop after SETM causes Measure mode to start. Only one sweep is taken and machine returns to software control.
4304 LCWORD	Load control word from AC. Bits of control word are defined below.
6314 OVSK	Skip on Measure memory overflow. Before entering Measure, the OVSK command will clear the overflow flag. The result of skipping or non-skipping should be ignored. Following a Measure sweep or group of sweeps, OVSK will indicate whether, during addition, any memory location became greater than 15/16 full.
4102 PULSE1	Pulse out rear panel BNC jack, marked PULSE1 or J6, 400 nsec long.
4104 PULSE2	Pulse out rear panel BNC connector marked PULSE 2 or J5, 400 nsec long.
6112 SENSE1	Skip if input to connector marked SENSE1 or J8 is high
6114 SENSE2	Skip if input to connector marked SENSE2 or J7 is high.

CONTROL WORD BIT ASSIGNMENTS

Bit #	Function Name	Bit = 1	Bit = Ø
Ø	Measure Add	Add data	Subtract data
1	Address Advance	Internal	External
2	Trigger	Positive Slope	Negative Slope
3	Recur	Auto Recur	Triggered Sweep
4	View	View Memory	View Input Signal
5	Continuous (overrides bit 4)	Continuous Display	Bit 4 in control
6	Not used		
7	Readout Light	On	Off
8	Compute Light	On	Off
9	Enable Clock for DWSK	On	Off
10	Digitizer Resolution)	00 = 12 bit 10 = 8 bit	Active only when front
11	Digitizer Resolution)	01 = 10 bit 11 = 6 bit	panel switch is set to Computer Control
12	Dual Input	Dual	Single
13	View Input A	View A	If both bits 13 & 14 are high,
14	View Input B	View B	both inputs will be shown overlapped
15	Transient Recorder	Transient Recorder bits 3 & 5 must be Ø	Normal
16	Homodecoupling Mode	Starts dwell signal running. Trigger only at dwell times (must be in Auto Recur mode)	Normal

APPENDIX I. ASCII Character Codes

The following list contains the 8-bit octal codes produced by standard ASR-33 Teletypes. This code is known as ASCII (American Standard Code for Information Interchange). The packed 6-bit code on the right is that used by the Assembler when storing text.

Several things should be noted about this code:

(a) The integers are biased by 260.
(b) The alphabet starts at 301.
(c) Most non-printing characters are less than 240.
(d) The CTRL key removes bit 6 from whatever key is typed: E = 305, CTRL/E = 205.
(e) The Shift key adds bit 4 to whatever key is typed: N = 316, SHIFT/N =   = 336.
(f) Leader-Trailer (200) tape can be generated by holding down SHIFT, CTRL, REPT and P. Release in opposite order.
(g) The characters [and] are generated by SHIFT/K and SHIFT/M respectively.

ASCII CHARACTER CODES

Character	Code	Packed 6 Bit
A	301	41
B	302	42
C	303	43
D	304	44
E	305	45
F	306	46
G	307	47
H	310	50
I	311	51
J	312	52
K	313	53
L	314	54
M	315	55
N	316	56
O	317	57
P	320	60
Q	321	61
R	322	62
S	323	63
T	324	64
U	325	65
V	326	66
W	327	67
X	330	70
Y	331	71
Z	332	72
0	260	20
1	261	21
2	262	22
3	263	23
4	264	24
5	265	25
6	266	26
7	267	27
8	270	30
9	271	31

Character	Code	Packed 6 Bit
!	241	01
"	242	02
#	243	03
$	244	04
%	245	05
&	246	06
'	247	07
(	250	10
)	251	11
*	252	12
+	253	13
,	254	14
-	255	15
.	256	16
/	257	17
:	272	32
;	273	33
<	274	34
=	275	35
>	276	36
?	277	37
@	300	40
[	333	73
\	334	74
]	335	75
↑	336	76
←	337	--
EOT	204	--
WRU	205	--
RU	206	--
BELL	207	--
TAB	211	--
Line Feed	212	--
FORM	214	--
Return	215	--
Space	240	00
ALT MODE	375	--
Rub Out	377	--

APPENDIX II

BIT ASSIGNMENTS

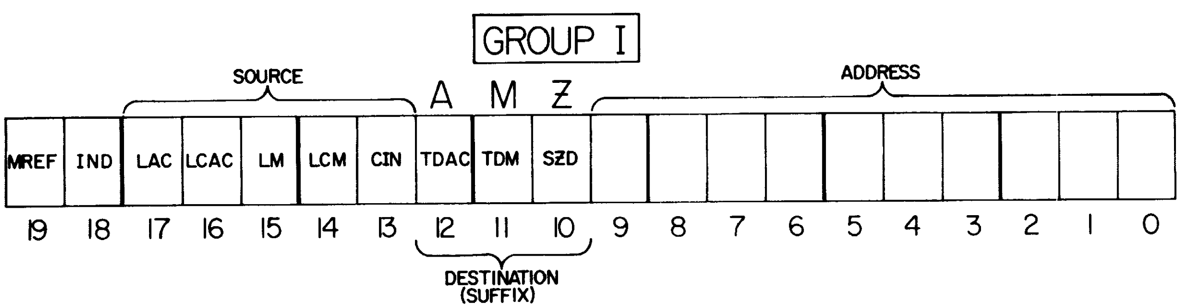
I. GROUP I INSTRUCTIONS

Group I instructions are actually combinations of five extremely simple machine instructions indicated in bits 13-17. These instructions are:

Bit	Operation	
17	LAC	Load the AC into the arithmetic unit
16	LCAC	Load the complement of the AC into the arithmetic unit
15	LM	Load the memory location specified by the addressing mode and bits 0-9 into the arithmetic unit
14	LCM	Load the complement of memory into the arithmetic unit
13	CIN	Increment the arithmetic unit contents

Those five source instructions are combined with the three destination instructions (suffixes):

12	TDAC	Transfer data to AC (A)
11	TDM	Transfer data to memory (M)
10	SZD	Skip on zero data (Z)



The instruction MEMA can thus be decomposed to LM and TDAC, or load memory into the arithmetic unit and transfer this data from the arithmetic unit to the AC. If several of the load instructions are combined, they are summed in the arithmetic unit. Thus A+MA is performed by LAC and LM, which causes the summation of the AC and memory, followed by TDAC which places this sum in the AC.

Similarly MPOMZ is decomposed into LM and CIN and TDM and SZD, which means load memory and increment it and then transfer the result back to memory and into the zero test register.

Subtraction of one from a number is accomplished by adding the number to -1 in the arithmetic unit. The minus one is created by loading a value and its complement, which produces 3777777, or -1. For instance MMOA is accomplished by LM + LAC + LCAC + TDAC and AMOZ into LAC + LM + LCM + SZD. Note that the actual contents of the register used to create the -1 are irrelevant since any number and its complement sum to become -1.

Negative numbers, you will recall, are formed by taking the one's complement and adding one. This is exemplified by ANGA (negative of AC to AC), which is decomposed into LCAC + CIN + TDAC. Similarly A-MM is broken into LAC + LCM + CIN + TDM.

The constant ONEA is constructed by CIN + TDAC and the constant ZERM by LM + LCM + CIN + TDM where LM and LCM create a -1 and CIN increments it to zero. Here it can be realized that ZERA (LM + LCM + CIN + TDAC) would also complement the link since the creation of the zero causes an arithmetic overflow.

II. TEST INSTRUCTIONS

The bits 13-17 are also active during test instructions. They are used to create some extra test conditions. The five basic test conditions are:

Bit	Condition	
4	ZDB	Zero data bus. The data bus is the output of the arithmetic unit.
3	ACØ	AC bit Ø
2	AC19	AC bit 19
1	COUT	Carry out of arithmetic unit
Ø	L	Link

These can be combined with bits 13-17 to produce the ZAC, MOAC and POAC test conditions. For instance to test the AC for +1 (POAC) we load the AC, add -1 to it and test for zero. This is accomplished by LAC + LM + LCM and either SKIP or EXCT on ZDB. Conversely the test for MOAC is accomplished by LAC + CIN + test for ZDB.

The COUT instruction has been found to be of limited use since whenever an arithmetic overflow occurs, the condition ZDB is probably also satisfied. The arithmetic carry out must be from the same instruction and not from a previous one, so that only the AC is generally known.

It is clearly also possible to combine the elementary test instructions. Thus SKIP AC19 L will produce a skip if AC19 = 1 <u>or</u> if the link = 1. Similarly SKIP AC19 ZAC will produce a skip if the AC is negative or zero.

Note in particular that combinations of ZAC and POAC or ZAC and MOAC will not produce the desired result since the actual skip is generated by the ZDB bit in either case. It is permissible to make the following combinations, however:

ZAC AC19 L ACØ — All possible combinations

MOAC AC19 ACØ L — All possible combinations

POAC AC19 ACØ L — All possible combinations

However the combinations

ZAC POAC

ZAC MOAC

are forbidden.

Further, it is not possible to combine EXCT and SKIP in a single instruction, since this implies that bit 5, the REVerse sensing bit is both on and off.

The bit assignments for these functions are shown below:

TEST AND MISC INSTRUCTIONS

0		LAC	LCAC	LM	LCM	CIN		1		1	0	1 MISC 0	0 TEST 1	REV	STOP ZDB	CLL ACØ	STL AC19	TLAC COUT	TACL L
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

TOP ROW IF MISC = 1
BOTTOM ROW IF TEST = 1

SKIP IF REV=0 TEST=1

EXCT IF REV=1 TEST=1

III. DISPLAY INSTRUCTIONS

The shift instructions take the number of shifts to be performed from bits Ø–3 unless bit 15 is set. If bit 15 is set the number of shifts is taken from the Vertical Display Scale switch. If the switch is set at 131K, no shifts are performed and if it is set at 4, 15_{10} shifts are performed. Any of the five types of shifts can be performed in this way, although the most useful one is the LASH, which has been given the special Assembler mnemonic VDSH. The octal codes for all of the shifts are:

From Bits 0-3		From Vertical Display Scale
5000	LASH	105000 (VDSH)
5020	RASH	105020
5040	LLSH	105040
5060	RLSH	105060
405020	RISH	505020

The TACXD instruction, 214001, is composed of LCAC + TDAC + 4001 so that the AC is complemented automatically before its transfer to the X-display register. This is because the X-display register requires the complemented value; but note that the original value in the AC is replaced by its complement after the instruction is executed.

The X-display can be initialized with 0 in a single instruction by using a combination of bits which produces a -1 (the complement of 0), such as 614001 which is LAC + LCAC + TDAC + 4001. Similarly, the X-display can be set to -1 by generating a 0 in the accumulator using bit 14 to clear the AC, resulting in the instruction 44001. The latter initialization is useful in display loops which use the INCXD instruction.

Finally, the Horizontal Display Scale switch is active during software control if the X-axis is advanced using the INCXD instruction. Thus 1024 INCXD's cause a full scale display if the switch is set to 1K Horizontal Display, 2048 are full scale at 2K and so forth. This switch does not influence the display if the X-axis is loaded with numbers using TACXD.

APPENDIX III

MODIFYING THE ASSEMBLER-EDITOR 1973

NIC 16-30417

The following modifications may be desirable for certain advanced users and can be easily accomplished from the switch register or by using Nicobug.

1. Changing the size of data storage. The locations SIZE and LLIMIT define the storage area for text as two 4096 word data stacks starting at address 100000. These locations are located as follows:

	Address	Contents	
LLIMIT,	2342	100000	/STARTING ADDRESS
SIZE,	2343	20000	/ONE MEMORY STACK

2. Changing the Append Command to echo at the Teletype. This can be used to generate error free source tapes or as an alternative method to Insert for adding new code.

Change location 3677 from	2024403	ONEM SUPSWT
to	2164403	ZERM SUPSWT

3. Changing the High Speed Reader-Punch I/O device codes. High speed equipment installed by users may utilize different I/O codes than those used by NIC. These are located as follows:

4565	44463	RHSR
4566	6464	HSRF
4522	6474	HSPF
4523	4473	PHSP

APPENDIX IV

POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

APPENDIX V

DECIMAL-OCTAL CONVERSION TABLE (Modulo 4096_{10})

	0	1	2	3	4	5	6	7	8	9
0000	0000	0001	0002	0003	0004	0005	0006	0007	0010	0011
0010	0012	0013	0014	0015	0016	0017	0020	0021	0022	0023
0020	0024	0025	0026	0027	0030	0031	0032	0033	0034	0035
0030	0036	0037	0040	0041	0042	0043	0044	0045	0046	0047
0040	0050	0051	0052	0053	0054	0055	0056	0057	0060	0061
0050	0062	0063	0064	0065	0066	0067	0070	0071	0072	0073
0060	0074	0075	0076	0077	0100	0101	0102	0103	0104	0105
0070	0106	0107	0110	0111	0112	0113	0114	0115	0116	0117
0080	0120	0121	0122	0123	0124	0125	0126	0127	0130	0131
0090	0132	0133	0134	0135	0136	0137	0140	0141	0142	0143
0100	0144	0145	0146	0147	0150	0151	0152	0153	0154	0155
0110	0156	0157	0160	0161	0162	0163	0164	0165	0166	0167
0120	0170	0171	0172	0173	0174	0175	0176	0177	0200	0201
0130	0202	0203	0204	0205	0206	0207	0210	0211	0212	0213
0140	0214	0215	0216	0217	0220	0221	0222	0223	0224	0225
0150	0226	0227	0230	0231	0232	0233	0234	0235	0236	0237
0160	0240	0241	0242	0243	0244	0245	0246	0247	0250	0251
0170	0252	0253	0254	0255	0256	0257	0260	0261	0262	0263
0180	0264	0265	0266	0267	0270	0271	0272	0273	0274	0275
0190	0276	0277	0300	0301	0302	0303	0304	0305	0306	0307
0200	0310	0311	0312	0313	0314	0315	0316	0317	0320	0321
0210	0322	0323	0324	0325	0326	0327	0330	0331	0332	0333
0220	0334	0335	0336	0337	0340	0341	0342	0343	0344	0345
0230	0346	0347	0350	0351	0352	0353	0354	0355	0356	0357
0240	0360	0361	0362	0363	0364	0365	0366	0367	0370	0371
0250	0372	0373	0374	0375	0376	0377	0400	0401	0402	0403
0260	0404	0405	0406	0407	0410	0411	0412	0413	0414	0415
0270	0416	0417	0420	0421	0422	0423	0424	0425	0426	0427
0280	0430	0431	0432	0433	0434	0435	0436	0437	0440	0441
0290	0442	0443	0444	0445	0446	0447	0450	0451	0452	0453
0300	0454	0455	0456	0457	0460	0461	0462	0463	0464	0465
0310	0466	0467	0470	0471	0472	0473	0474	0475	0476	0477
0320	0500	0501	0502	0503	0504	0505	0506	0507	0510	0511
0330	0512	0513	0514	0515	0516	0517	0520	0521	0522	0523
0340	0524	0525	0526	0527	0530	0531	0532	0533	0534	0535
0350	0536	0537	0540	0541	0542	0543	0544	0545	0546	0547
0360	0550	0551	0552	0553	0554	0555	0556	0557	0560	0561
0370	0562	0563	0564	0565	0566	0567	0570	0571	0572	0573
0380	0574	0575	0576	0577	0600	0601	0602	0603	0604	0605
0390	0606	0607	0610	0611	0612	0613	0614	0615	0616	0617
0400	0620	0621	0622	0623	0624	0625	0626	0627	0630	0631
0410	0632	0633	0634	0635	0636	0637	0640	0641	0642	0643
0420	0644	0645	0646	0647	0650	0651	0652	0653	0654	0655
0430	0656	0657	0660	0661	0662	0663	0664	0665	0666	0667
0440	0670	0671	0672	0673	0674	0675	0676	0677	0700	0701
0450	0702	0703	0704	0705	0706	0707	0710	0711	0712	0713
0460	0714	0715	0716	0717	0720	0721	0722	0723	0724	0725
0470	0726	0727	0730	0731	0732	0733	0734	0735	0736	0737
0480	0740	0741	0742	0743	0744	0745	0746	0747	0750	0751
0490	0752	0753	0754	0755	0756	0757	0760	0761	0762	0763

	0	1	2	3	4	5	6	7	8	9
0500	0764	0765	0766	0767	0770	0771	0772	0773	0774	0775
0510	0776	0777	1000	1001	1002	1003	1004	1005	1006	1007
0520	1010	1011	1012	1013	1014	1015	1016	1017	1020	1021
0530	1022	1023	1024	1025	1026	1027	1030	1031	1032	1033
0540	1034	1035	1036	1037	1040	1041	1042	1043	1044	1045
0550	1046	1047	1050	1051	1052	1053	1054	1055	1056	1057
0560	1060	1061	1062	1063	1064	1065	1066	1067	1070	1071
0570	1072	1073	1074	1075	1076	1077	1100	1101	1102	1103
0580	1104	1105	1106	1107	1110	1111	1112	1113	1114	1115
0590	1116	1117	1120	1121	1122	1123	1124	1125	1126	1127
0600	1130	1131	1132	1133	1134	1135	1136	1137	1140	1141
0610	1142	1143	1144	1145	1146	1147	1150	1151	1152	1153
0620	1154	1155	1156	1157	1160	1161	1162	1163	1164	1165
0630	1166	1167	1170	1171	1172	1173	1174	1175	1176	1177
0640	1200	1201	1202	1203	1204	1205	1206	1207	1210	1211
0650	1212	1213	1214	1215	1216	1217	1220	1221	1222	1223
0660	1224	1225	1226	1227	1230	1231	1232	1233	1234	1235
0670	1236	1237	1240	1241	1242	1243	1244	1245	1246	1247
0680	1250	1251	1252	1253	1254	1255	1256	1257	1260	1261
0690	1262	1263	1264	1265	1266	1267	1270	1271	1272	1273
0700	1274	1275	1276	1277	1300	1301	1302	1303	1304	1305
0710	1306	1307	1310	1311	1312	1313	1314	1315	1316	1317
0720	1320	1321	1322	1323	1324	1325	1326	1327	1330	1331
0730	1332	1333	1334	1335	1336	1337	1340	1341	1342	1343
0740	1344	1345	1346	1347	1350	1351	1352	1353	1354	1355
0750	1356	1357	1360	1361	1362	1363	1364	1365	1366	1367
0760	1370	1371	1372	1373	1374	1375	1376	1377	1400	1401
0770	1402	1403	1404	1405	1406	1407	1410	1411	1412	1413
0780	1414	1415	1416	1417	1420	1421	1422	1423	1424	1425
0790	1426	1427	1430	1431	1432	1433	1434	1435	1436	1437
0800	1440	1441	1442	1443	1444	1445	1446	1447	1450	1451
0810	1452	1453	1454	1455	1456	1457	1460	1461	1462	1463
0820	1464	1465	1466	1467	1470	1471	1472	1473	1474	1475
0830	1476	1477	1500	1501	1502	1503	1504	1505	1506	1507
0840	1510	1511	1512	1513	1514	1515	1516	1517	1520	1521
0850	1522	1523	1524	1525	1526	1527	1530	1531	1532	1533
0860	1534	1535	1536	1537	1540	1541	1542	1543	1544	1545
0870	1546	1547	1550	1551	1552	1553	1554	1555	1556	1557
0880	1560	1561	1562	1563	1564	1565	1566	1567	1570	1571
0890	1572	1573	1574	1575	1576	1577	1600	1601	1602	1603
0900	1604	1605	1606	1607	1610	1611	1612	1613	1614	1615
0910	1616	1617	1620	1621	1622	1623	1624	1625	1626	1627
0920	1630	1631	1632	1633	1634	1635	1636	1637	1640	1641
0930	1642	1643	1644	1645	1646	1647	1650	1651	1652	1653
0940	1654	1655	1656	1657	1660	1661	1662	1663	1664	1665
0950	1666	1667	1670	1671	1672	1673	1674	1675	1676	1677
0960	1700	1701	1702	1703	1704	1705	1706	1707	1710	1711
0970	1712	1713	1714	1715	1716	1717	1720	1721	1722	1723
0980	1724	1725	1726	1727	1730	1731	1732	1733	1734	1735
0990	1736	1737	1740	1741	1742	1743	1744	1745	1746	1747

	0	1	2	3	4	5	6	7	8	9
1000	1750	1751	1752	1753	1754	1755	1756	1757	1760	1761
1010	1762	1763	1764	1765	1766	1767	1770	1771	1772	1773
1020	1774	1775	1776	1777	2000	2001	2002	2003	2004	2005
1030	2006	2007	2010	2011	2012	2013	2014	2015	2016	2017
1040	2020	2021	2022	2023	2024	2025	2026	2027	2030	2031
1050	2032	2033	2034	2035	2036	2037	2040	2041	2042	2043
1060	2044	2045	2046	2047	2050	2051	2052	2053	2054	2055
1070	2056	2057	2060	2061	2062	2063	2064	2065	2066	2067
1080	2070	2071	2072	2073	2074	2075	2076	2077	2100	2101
1090	2102	2103	2104	2105	2106	2107	2110	2111	2112	2113
1100	2114	2115	2116	2117	2120	2121	2122	2123	2124	2125
1110	2126	2127	2130	2131	2132	2133	2134	2135	2136	2137
1120	2140	2141	2142	2143	2144	2145	2146	2147	2150	2151
1130	2152	2153	2154	2155	2156	2157	2160	2161	2162	2163
1140	2164	2165	2166	2167	2170	2171	2172	2173	2174	2175
1150	2176	2177	2200	2201	2202	2203	2204	2205	2206	2207
1160	2210	2211	2212	2213	2214	2215	2216	2217	2220	2221
1170	2222	2223	2224	2225	2226	2227	2230	2231	2232	2233
1180	2234	2235	2236	2237	2240	2241	2242	2243	2244	2245
1190	2246	2247	2250	2251	2252	2253	2254	2255	2256	2257
1200	2260	2261	2262	2263	2264	2265	2266	2267	2270	2271
1210	2272	2273	2274	2275	2276	2277	2300	2301	2302	2303
1220	2304	2305	2306	2307	2310	2311	2312	2313	2314	2315
1230	2316	2317	2320	2321	2322	2323	2324	2325	2326	2327
1240	2330	2331	2332	2333	2334	2335	2336	2337	2340	2341
1250	2342	2343	2344	2345	2346	2347	2350	2351	2352	2353
1260	2354	2355	2356	2357	2360	2361	2362	2363	2364	2365
1270	2366	2367	2370	2371	2372	2373	2374	2375	2376	2377
1280	2400	2401	2402	2403	2404	2405	2406	2407	2410	2411
1290	2412	2413	2414	2415	2416	2417	2420	2421	2422	2423
1300	2424	2425	2426	2427	2430	2431	2432	2433	2434	2435
1310	2436	2437	2440	2441	2442	2443	2444	2445	2446	2447
1320	2450	2451	2452	2453	2454	2455	2456	2457	2460	2461
1330	2462	2463	2464	2465	2466	2467	2470	2471	2472	2473
1340	2474	2475	2476	2477	2500	2501	2502	2503	2504	2505
1350	2506	2507	2510	2511	2512	2513	2514	2515	2516	2517
1360	2520	2521	2522	2523	2524	2525	2526	2527	2530	2531
1370	2532	2533	2534	2535	2536	2537	2540	2541	2542	2543
1380	2544	2545	2546	2547	2550	2551	2552	2553	2554	2555
1390	2556	2557	2560	2561	2562	2563	2564	2565	2566	2567
1400	2570	2571	2572	2573	2574	2575	2576	2577	2600	2601
1410	2602	2603	2604	2605	2606	2607	2610	2611	2612	2613
1420	2614	2615	2616	2617	2620	2621	2622	2623	2624	2625
1430	2626	2627	2630	2631	2632	2633	2634	2635	2636	2637
1440	2640	2641	2642	2643	2644	2645	2646	2647	2650	2651
1450	2652	2653	2654	2655	2656	2657	2660	2661	2662	2663
1460	2664	2665	2666	2667	2670	2671	2672	2673	2674	2675
1470	2676	2677	2700	2701	2702	2703	2704	2705	2706	2707
1480	2710	2711	2712	2713	2714	2715	2716	2717	2720	2721
1490	2722	2723	2724	2725	2726	2727	2730	2731	2732	2733

	0	1	2	3	4	5	6	7	8	9
1500	2734	2735	2736	2737	2740	2741	2742	2743	2744	2745
1510	2746	2747	2750	2751	2752	2753	2754	2755	2756	2757
1520	2760	2761	2762	2763	2764	2765	2766	2767	2770	2771
1530	2772	2773	2774	2775	2776	2777	3000	3001	3002	3003
1540	3004	3005	3006	3007	3010	3011	3012	3013	3014	3015
1550	3016	3017	3020	3021	3022	3023	3024	3025	3026	3027
1560	3030	3031	3032	3033	3034	3035	3036	3037	3040	3041
1570	3042	3043	3044	3045	3046	3047	3050	3051	3052	3053
1580	3054	3055	3056	3057	3060	3061	3062	3063	3064	3065
1590	3066	3067	3070	3071	3072	3073	3074	3075	3076	3077
1600	3100	3101	3102	3103	3104	3105	3106	3107	3110	3111
1610	3112	3113	3114	3115	3116	3117	3120	3121	3122	3123
1620	3124	3125	3126	3127	3130	3131	3132	3133	3134	3135
1630	3136	3137	3140	3141	3142	3143	3144	3145	3146	3147
1640	3150	3151	3152	3153	3154	3155	3156	3157	3160	3161
1650	3162	3163	3164	3165	3166	3167	3170	3171	3172	3173
1660	3174	3175	3176	3177	3200	3201	3202	3203	3204	3205
1670	3206	3207	3210	3211	3212	3213	3214	3215	3216	3217
1680	3220	3221	3222	3223	3224	3225	3226	3227	3230	3231
1690	3232	3233	3234	3235	3236	3237	3240	3241	3242	3243
1700	3244	3245	3246	3247	3250	3251	3252	3253	3254	3255
1710	3256	3257	3260	3261	3262	3263	3264	3265	3266	3267
1720	3270	3271	3272	3273	3274	3275	3276	3277	3300	3301
1730	3302	3303	3304	3305	3306	3307	3310	3311	3312	3313
1740	3314	3315	3316	3317	3320	3321	3322	3323	3324	3325
1750	3326	3327	3330	3331	3332	3333	3334	3335	3336	3337
1760	3340	3341	3342	3343	3344	3345	3346	3347	3350	3351
1770	3352	3353	3354	3355	3356	3357	3360	3361	3362	3363
1780	3364	3365	3366	3367	3370	3371	3372	3373	3374	3375
1790	3376	3377	3400	3401	3402	3403	3404	3405	3406	3407
1800	3410	3411	3412	3413	3414	3415	3416	3417	3420	3421
1810	3422	3423	3424	3425	3426	3427	3430	3431	3432	3433
1820	3434	3435	3436	3437	3440	3441	3442	3443	3444	3445
1830	3446	3447	3450	3451	3452	3453	3454	3455	3456	3457
1840	3460	3461	3462	3463	3464	3465	3466	3467	3470	3471
1850	3472	3473	3474	3475	3476	3477	3500	3501	3502	3503
1860	3504	3505	3506	3507	3510	3511	3512	3513	3514	3515
1870	3516	3517	3520	3521	3522	3523	3524	3525	3526	3527
1880	3530	3531	3532	3533	3534	3535	3536	3537	3540	3541
1890	3542	3543	3544	3545	3546	3547	3550	3551	3552	3553
1900	3554	3555	3556	3557	3560	3561	3562	3563	3564	3565
1910	3566	3567	3570	3571	3572	3573	3574	3575	3576	3577
1920	3600	3601	3602	3603	3604	3605	3606	3607	3610	3611
1930	3612	3613	3614	3615	3616	3617	3620	3621	3622	3623
1940	3624	3625	3626	3627	3630	3631	3632	3633	3634	3635
1950	3636	3637	3640	3641	3642	3643	3644	3645	3646	3647
1960	3650	3651	3652	3653	3654	3655	3656	3657	3660	3661
1970	3662	3663	3664	3665	3666	3667	3670	3671	3672	3673
1980	3674	3675	3676	3677	3700	3701	3702	3703	3704	3705
1990	3706	3707	3710	3711	3712	3713	3714	3715	3716	3717

	0	1	2	3	4	5	6	7	8	9
2000	3720	3721	3722	3723	3724	3725	3726	3727	3730	3731
2010	3732	3733	3734	3735	3736	3737	3740	3741	3742	3743
2020	3744	3745	3746	3747	3750	3751	3752	3753	3754	3755
2030	3756	3757	3760	3761	3762	3763	3764	3765	3766	3767
2040	3770	3771	3772	3773	3774	3775	3776	3777	4000	4001
2050	4002	4003	4004	4005	4006	4007	4010	4011	4012	4013
2060	4014	4015	4016	4017	4020	4021	4022	4023	4024	4025
2070	4026	4027	4030	4031	4032	4033	4034	4035	4036	4037
2080	4040	4041	4042	4043	4044	4045	4046	4047	4050	4051
2090	4052	4053	4054	4055	4056	4057	4060	4061	4062	4063
2100	4064	4065	4066	4067	4070	4071	4072	4073	4074	4075
2110	4076	4077	4100	4101	4102	4103	4104	4105	4106	4107
2120	4110	4111	4112	4113	4114	4115	4116	4117	4120	4121
2130	4122	4123	4124	4125	4126	4127	4130	4131	4132	4133
2140	4134	4135	4136	4137	4140	4141	4142	4143	4144	4145
2150	4146	4147	4150	4151	4152	4153	4154	4155	4156	4157
2160	4160	4161	4162	4163	4164	4165	4166	4167	4170	4171
2170	4172	4173	4174	4175	4176	4177	4200	4201	4202	4203
2180	4204	4205	4206	4207	4210	4211	4212	4213	4214	4215
2190	4216	4217	4220	4221	4222	4223	4224	4225	4226	4227
2200	4230	4231	4232	4233	4234	4235	4236	4237	4240	4241
2210	4242	4243	4244	4245	4246	4247	4250	4251	4252	4253
2220	4254	4255	4256	4257	4260	4261	4262	4263	4264	4265
2230	4266	4267	4270	4271	4272	4273	4274	4275	4276	4277
2240	4300	4301	4302	4303	4304	4305	4306	4307	4310	4311
2250	4312	4313	4314	4315	4316	4317	4320	4321	4322	4323
2260	4324	4325	4326	4327	4330	4331	4332	4333	4334	4335
2270	4336	4337	4340	4341	4342	4343	4344	4345	4346	4347
2280	4350	4351	4352	4353	4354	4355	4356	4357	4360	4361
2290	4362	4363	4364	4365	4366	4367	4370	4371	4372	4373
2300	4374	4375	4376	4377	4400	4401	4402	4403	4404	4405
2310	4406	4407	4410	4411	4412	4413	4414	4415	4416	4417
2320	4420	4421	4422	4423	4424	4425	4426	4427	4430	4431
2330	4432	4433	4434	4435	4436	4437	4440	4441	4442	4443
2340	4444	4445	4446	4447	4450	4451	4452	4453	4454	4455
2350	4456	4457	4460	4461	4462	4463	4464	4465	4466	4467
2360	4470	4471	4472	4473	4474	4475	4476	4477	4500	4501
2370	4502	4503	4504	4505	4506	4507	4510	4511	4512	4513
2380	4514	4515	4516	4517	4520	4521	4522	4523	4524	4525
2390	4526	4527	4530	4531	4532	4533	4534	4535	4536	4537
2400	4540	4541	4542	4543	4544	4545	4546	4547	4550	4551
2410	4552	4553	4554	4555	4556	4557	4560	4561	4562	4563
2420	4564	4565	4566	4567	4570	4571	4572	4573	4574	4575
2430	4576	4577	4600	4601	4602	4603	4604	4605	4606	4607
2440	4610	4611	4612	4613	4614	4615	4616	4617	4620	4621
2450	4622	4623	4624	4625	4626	4627	4630	4631	4632	4633
2460	4634	4635	4636	4637	4640	4641	4642	4643	4644	4645
2470	4646	4647	4650	4651	4652	4653	4654	4655	4656	4657
2480	4660	4661	4662	4663	4664	4665	4666	4667	4670	4671
2490	4672	4673	4674	4675	4676	4677	4700	4701	4702	4703

	0	1	2	3	4	5	6	7	8	9
2500	4704	4705	4706	4707	4710	4711	4712	4713	4714	4715
2510	4716	4717	4720	4721	4722	4723	4724	4725	4726	4727
2520	4730	4731	4732	4733	4734	4735	4736	4737	4740	4741
2530	4742	4743	4744	4745	4746	4747	4750	4751	4752	4753
2540	4754	4755	4756	4757	4760	4761	4762	4763	4764	4765
2550	4766	4767	4770	4771	4772	4773	4774	4775	4776	4777
2560	5000	5001	5002	5003	5004	5005	5006	5007	5010	5011
2570	5012	5013	5014	5015	5016	5017	5020	5021	5022	5023
2580	5024	5025	5026	5027	5030	5031	5032	5033	5034	5035
2590	5036	5037	5040	5041	5042	5043	5044	5045	5046	5047
2600	5050	5051	5052	5053	5054	5055	5056	5057	5060	5061
2610	5062	5063	5064	5065	5066	5067	5070	5071	5072	5073
2620	5074	5075	5076	5077	5100	5101	5102	5103	5104	5105
2630	5106	5107	5110	5111	5112	5113	5114	5115	5116	5117
2640	5120	5121	5122	5123	5124	5125	5126	5127	5130	5131
2650	5132	5133	5134	5135	5136	5137	5140	5141	5142	5143
2660	5144	5145	5146	5147	5150	5151	5152	5153	5154	5155
2670	5156	5157	5160	5161	5162	5163	5164	5165	5166	5167
2680	5170	5171	5172	5173	5174	5175	5176	5177	5200	5201
2690	5202	5203	5204	5205	5206	5207	5210	5211	5212	5213
2700	5214	5215	5216	5217	5220	5221	5222	5223	5224	5225
2710	5226	5227	5230	5231	5232	5233	5234	5235	5236	5237
2720	5240	5241	5242	5243	5244	5245	5246	5247	5250	5251
2730	5252	5253	5254	5255	5256	5257	5260	5261	5262	5263
2740	5264	5265	5266	5267	5270	5271	5272	5273	5274	5275
2750	5276	5277	5300	5301	5302	5303	5304	5305	5306	5307
2760	5310	5311	5312	5313	5314	5315	5316	5317	5320	5321
2770	5322	5323	5324	5325	5326	5327	5330	5331	5332	5333
2780	5334	5335	5336	5337	5340	5341	5342	5343	5344	5345
2790	5346	5347	5350	5351	5352	5353	5354	5355	5356	5357
2800	5360	5361	5362	5363	5364	5365	5366	5367	5370	5371
2810	5372	5373	5374	5375	5376	5377	5400	5401	5402	5403
2820	5404	5405	5406	5407	5410	5411	5412	5413	5414	5415
2830	5416	5417	5420	5421	5422	5423	5424	5425	5426	5427
2840	5430	5431	5432	5433	5434	5435	5436	5437	5440	5441
2850	5442	5443	5444	5445	5446	5447	5450	5451	5452	5453
2860	5454	5455	5456	5457	5460	5461	5462	5463	5464	5465
2870	5466	5467	5470	5471	5472	5473	5474	5475	5476	5477
2880	5500	5501	5502	5503	5504	5505	5506	5507	5510	5511
2890	5512	5513	5514	5515	5516	5517	5520	5521	5522	5523
2900	5524	5525	5526	5527	5530	5531	5532	5533	5534	5535
2910	5536	5537	5540	5541	5542	5543	5544	5545	5546	5547
2920	5550	5551	5552	5553	5554	5555	5556	5557	5560	5561
2930	5562	5563	5564	5565	5566	5567	5570	5571	5572	5573
2940	5574	5575	5576	5577	5600	5601	5602	5603	5604	5605
2950	5606	5607	5610	5611	5612	5613	5614	5615	5616	5617
2960	5620	5621	5622	5623	5624	5625	5626	5627	5630	5631
2970	5632	5633	5634	5635	5636	5637	5640	5641	5642	5643
2980	5644	5645	5646	5647	5650	5651	5652	5653	5654	5655
2990	5656	5657	5660	5661	5662	5663	5664	5665	5666	5667

	0	1	2	3	4	5	6	7	8	9
3000	5670	5671	5672	5673	5674	5675	5676	5677	5700	5701
3010	5702	5703	5704	5705	5706	5707	5710	5711	5712	5713
3020	5714	5715	5716	5717	5720	5721	5722	5723	5724	5725
3030	5726	5727	5730	5731	5732	5733	5734	5735	5736	5737
3040	5740	5741	5742	5743	5744	5745	5746	5747	5750	5751
3050	5752	5753	5754	5755	5756	5757	5760	5761	5762	5763
3060	5764	5765	5766	5767	5770	5771	5772	5773	5774	5775
3070	5776	5777	6000	6001	6002	6003	6004	6005	6006	6007
3080	6010	6011	6012	6013	6014	6015	6016	6017	6020	6021
3090	6022	6023	6024	6025	6026	6027	6030	6031	6032	6033
3100	6034	6035	6036	6037	6040	6041	6042	6043	6044	6045
3110	6046	6047	6050	6051	6052	6053	6054	6055	6056	6057
3120	6060	6061	6062	6063	6064	6065	6066	6067	6070	6071
3130	6072	6073	6074	6075	6076	6077	6100	6101	6102	6103
3140	6104	6105	6106	6107	6110	6111	6112	6113	6114	6115
3150	6116	6117	6120	6121	6122	6123	6124	6125	6126	6127
3160	6130	6131	6132	6133	6134	6135	6136	6137	6140	6141
3170	6142	6143	6144	6145	6146	6147	6150	6151	6152	6153
3180	6154	6155	6156	6157	6160	6161	6162	6163	6164	6165
3190	6166	6167	6170	6171	6172	6173	6174	6175	6176	6177
3200	6200	6201	6202	6203	6204	6205	6206	6207	6210	6211
3210	6212	6213	6214	6215	6216	6217	6220	6221	6222	6223
3220	6224	6225	6226	6227	6230	6231	6232	6233	6234	6235
3230	6236	6237	6240	6241	6242	6243	6244	6245	6246	6247
3240	6250	6251	6252	6253	6254	6255	6256	6257	6260	6261
3250	6262	6263	6264	6265	6266	6267	6270	6271	6272	6273
3260	6274	6275	6276	6277	6300	6301	6302	6303	6304	6305
3270	6306	6307	6310	6311	6312	6313	6314	6315	6316	6317
3280	6320	6321	6322	6323	6324	6325	6326	6327	6330	6331
3290	6332	6333	6334	6335	6336	6337	6340	6341	6342	6343
3300	6344	6345	6346	6347	6350	6351	6352	6353	6354	6355
3310	6356	6357	6360	6361	6362	6363	6364	6365	6366	6367
3320	6370	6371	6372	6373	6374	6375	6376	6377	6400	6401
3330	6402	6403	6404	6405	6406	6407	6410	6411	6412	6413
3340	6414	6415	6416	6417	6420	6421	6422	6423	6424	6425
3350	6426	6427	6430	6431	6432	6433	6434	6435	6436	6437
3360	6440	6441	6442	6443	6444	6445	6446	6447	6450	6451
3370	6452	6453	6454	6455	6456	6457	6460	6461	6462	6463
3380	6464	6465	6466	6467	6470	6471	6472	6473	6474	6475
3390	6476	6477	6500	6501	6502	6503	6504	6505	6506	6507
3400	6510	6511	6512	6513	6514	6515	6516	6517	6520	6521
3410	6522	6523	6524	6525	6526	6527	6530	6531	6532	6533
3420	6534	6535	6536	6537	6540	6541	6542	6543	6544	6545
3430	6546	6547	6550	6551	6552	6553	6554	6555	6556	6557
3440	6560	6561	6562	6563	6564	6565	6566	6567	6570	6571
3450	6572	6573	6574	6575	6576	6577	6600	6601	6602	6603
3460	6604	6605	6606	6607	6610	6611	6612	6613	6614	6615
3470	6616	6617	6620	6621	6622	6623	6624	6625	6626	6627
3480	6630	6631	6632	6633	6634	6635	6636	6637	6640	6641
3490	6642	6643	6644	6645	6646	6647	6650	6651	6652	6653

	0	1	2	3	4	5	6	7	8	9
3500	6654	6655	6656	6657	6660	6661	6662	6663	6664	6665
3510	6666	6667	6670	6671	6672	6673	6674	6675	6676	6677
3520	6700	6701	6702	6703	6704	6705	6706	6707	6710	6711
3530	6712	6713	6714	6715	6716	6717	6720	6721	6722	6723
3540	6724	6725	6726	6727	6730	6731	6732	6733	6734	6735
3550	6736	6737	6740	6741	6742	6743	6744	6745	6746	6747
3560	6750	6751	6752	6753	6754	6755	6756	6757	6760	6761
3570	6762	6763	6764	6765	6766	6767	6770	6771	6772	6773
3580	6774	6775	6776	6777	7000	7001	7002	7003	7004	7005
3590	7006	7007	7010	7011	7012	7013	7014	7015	7016	7017
3600	7020	7021	7022	7023	7024	7025	7026	7027	7030	7031
3610	7032	7033	7034	7035	7036	7037	7040	7041	7042	7043
3620	7044	7045	7046	7047	7050	7051	7052	7053	7054	7055
3630	7056	7057	7060	7061	7062	7063	7064	7065	7066	7067
3640	7070	7071	7072	7073	7074	7075	7076	7077	7100	7101
3650	7102	7103	7104	7105	7106	7107	7110	7111	7112	7113
3660	7114	7115	7116	7117	7120	7121	7122	7123	7124	7125
3670	7126	7127	7130	7131	7132	7133	7134	7135	7136	7137
3680	7140	7141	7142	7143	7144	7145	7146	7147	7150	7151
3690	7152	7153	7154	7155	7156	7157	7160	7161	7162	7163
3700	7164	7165	7166	7167	7170	7171	7172	7173	7174	7175
3710	7176	7177	7200	7201	7202	7203	7204	7205	7206	7207
3720	7210	7211	7212	7213	7214	7215	7216	7217	7220	7221
3730	7222	7223	7224	7225	7226	7227	7230	7231	7232	7233
3740	7234	7235	7236	7237	7240	7241	7242	7243	7244	7245
3750	7246	7247	7250	7251	7252	7253	7254	7255	7256	7257
3760	7260	7261	7262	7263	7264	7265	7266	7267	7270	7271
3770	7272	7273	7274	7275	7276	7277	7300	7301	7302	7303
3780	7304	7305	7306	7307	7310	7311	7312	7313	7314	7315
3790	7316	7317	7320	7321	7322	7323	7324	7325	7326	7327
3800	7330	7331	7332	7333	7334	7335	7336	7337	7340	7341
3810	7342	7343	7344	7345	7346	7347	7350	7351	7352	7353
3820	7354	7355	7356	7357	7360	7361	7362	7363	7364	7365
3830	7366	7367	7370	7371	7372	7373	7374	7375	7376	7377
3840	7400	7401	7402	7403	7404	7405	7406	7407	7410	7411
3850	7412	7413	7414	7415	7416	7417	7420	7421	7422	7423
3860	7424	7425	7426	7427	7430	7431	7432	7433	7434	7435
3870	7436	7437	7440	7441	7442	7443	7444	7445	7446	7447
3880	7450	7451	7452	7453	7454	7455	7456	7457	7460	7461
3890	7462	7463	7464	7465	7466	7467	7470	7471	7472	7473
3900	7474	7475	7476	7477	7500	7501	7502	7503	7504	7505
3910	7506	7507	7510	7511	7512	7513	7514	7515	7516	7517
3920	7520	7521	7522	7523	7524	7525	7526	7527	7530	7531
3930	7532	7533	7534	7535	7536	7537	7540	7541	7542	7543
3940	7544	7545	7546	7547	7550	7551	7552	7553	7554	7555
3950	7556	7557	7560	7561	7562	7563	7564	7565	7566	7567
3960	7570	7571	7572	7573	7574	7575	7576	7577	7600	7601
3970	7602	7603	7604	7605	7606	7607	7610	7611	7612	7613
3980	7614	7615	7616	7617	7620	7621	7622	7623	7624	7625
3990	7626	7627	7630	7631	7632	7633	7634	7635	7636	7637

	0	1	2	3	4	5	6	7	8	9
4000	7640	7641	7642	7643	7644	7645	7646	7647	7650	7651
4010	7652	7653	7654	7655	7656	7657	7660	7661	7662	7663
4020	7664	7665	7666	7667	7670	7671	7672	7673	7674	7675
4030	7676	7677	7700	7701	7702	7703	7704	7705	7706	7707
4040	7710	7711	7712	7713	7714	7715	7716	7717	7720	7721
4050	7722	7723	7724	7725	7726	7727	7730	7731	7732	7733
4060	7734	7735	7736	7737	7740	7741	7742	7743	7744	7745
4070	7746	7747	7750	7751	7752	7753	7754	7755	7756	7757
4080	7760	7761	7762	7763	7764	7765	7766	7767	7770	7771
4090	7772	7773	7774	7775	7776	7777	0000	0001	0002	0003
4100	0004	0005	0006	0007	0010	0011	0012	0013	0014	0015
4110	0016	0017	0020	0021	0022	0023	0024	0025	0026	0027
4120	0030	0031	0032	0033	0034	0035	0036	0037	0040	0041
4130	0042	0043	0044	0045	0046	0047	0050	0051	0052	0053
4140	0054	0055	0056	0057	0060	0061	0062	0063	0064	0065
4150	0066	0067	0070	0071	0072	0073	0074	0075	0076	0077
4160	0100	0101	0102	0103	0104	0105	0106	0107	0110	0111
4170	0112	0113	0114	0115	0116	0117	0120	0121	0122	0123
4180	0124	0125	0126	0127	0130	0131	0132	0133	0134	0135
4190	0136	0137	0140	0141	0142	0143	0144	0145	0146	0147
4200	0150	0151	0152	0153	0154	0155	0156	0157	0160	0161
4210	0162	0163	0164	0165	0166	0167	0170	0171	0172	0173
4220	0174	0175	0176	0177	0200	0201	0202	0203	0204	0205
4230	0206	0207	0210	0211	0212	0213	0214	0215	0216	0217
4240	0220	0221	0222	0223	0224	0225	0226	0227	0230	0231
4250	0232	0233	0234	0235	0236	0237	0240	0241	0242	0243
4260	0244	0245	0246	0247	0250	0251	0252	0253	0254	0255
4270	0256	0257	0260	0261	0262	0263	0264	0265	0266	0267
4280	0270	0271	0272	0273	0274	0275	0276	0277	0300	0301
4290	0302	0303	0304	0305	0306	0307	0310	0311	0312	0313
4300	0314	0315	0316	0317	0320	0321	0322	0323	0324	0325
4310	0326	0327	0330	0331	0332	0333	0334	0335	0336	0337
4320	0340	0341	0342	0343	0344	0345	0346	0347	0350	0351
4330	0352	0353	0354	0355	0356	0357	0360	0361	0362	0363
4340	0364	0365	0366	0367	0370	0371	0372	0373	0374	0375
4350	0376	0377	0400	0401	0402	0403	0404	0405	0406	0407
4360	0410	0411	0412	0413	0414	0415	0416	0417	0420	0421
4370	0422	0423	0424	0425	0426	0427	0430	0431	0432	0433
4380	0434	0435	0436	0437	0440	0441	0442	0443	0444	0445
4390	0446	0447	0450	0451	0452	0453	0454	0455	0456	0457
4400	0460	0461	0462	0463	0464	0465	0466	0467	0470	0471
4410	0472	0473	0474	0475	0476	0477	0500	0501	0502	0503
4420	0504	0505	0506	0507	0510	0511	0512	0513	0514	0515
4430	0516	0517	0520	0521	0522	0523	0524	0525	0526	0527
4440	0530	0531	0532	0533	0534	0535	0536	0537	0540	0541
4450	0542	0543	0544	0545	0546	0547	0550	0551	0552	0553
4460	0554	0555	0556	0557	0560	0561	0562	0563	0564	0565
4470	0566	0567	0570	0571	0572	0573	0574	0575	0576	0577
4480	0600	0601	0602	0603	0604	0605	0606	0607	0610	0611
4490	0612	0613	0614	0615	0616	0617	0620	0621	0622	0623

Octal		Decimal
10000	=	4,096
100000	=	32,768
1000000	=	262,144
23,420	=	10,000
303,240	=	100,000
3,641,100	=	1,000,000

INDEX

NIC SALES OFFICES

CT, MA, ME, NH, RI, VT
Harding Bush
Nicolet Instrument Corporation
2120 Commonwealth Avenue
Auburndale, MA 02166
(617) 969-7420
TWX: 710-335-1954

Upper New York State
James Lappegaard
Nicolet Instrument Corporation
245 Livingston Street
Northvale, NJ 07647
(201) 767-7100
TWX: 710-991-9619

NY, NJ, PA
Bernard Conti
Nicolet Instrument Corporation
P. O. Box 36
Old Bethpage, NY 11804
(516) 249-6360

Wash.DC , MD, VA, WV,DE
James Cavanaugh
Nicolet Instrument Corporation
4620 Wisconsin Avenue, N. W.
Washington, DC 20016
(202) 686-0189

IL, IN, MO, OH, KY
Frank B. Contratto
Nicolet Instrument Corporation
500 East Higgins Road
Elk Grove, IL 60007
(312) 956-0404
TWX: 910-222-5999

IA, MI, MN, KS, ND, SD, NE, WI
Richard Bohn
Nicolet Instrument Corporation
5225 Verona Road
Madison, WI 53711
(608) 271-3333
TWX: 910-286-2713

NC, SC, GA, AL, TN, FL, MS
Dr. H. B. Evans
Nicolet Instrument Corporation
140 South Dekalb Office Park
3009 Rainbow Drive
Decatur, GA 30034
(404) 243-1219

TX, LA, OK, AR
Jerry A. Meyer
Nicolet Instrument Corporation
701 - 15th Street
Plano, TX 75074
(214) 424-8611

CO, NM, AZ, UT, MT, ID, WY
Bruce B. Lent
Nicolet Instrument Corporation
6023 South Lamar Drive
Littleton, CO. 80123
(303) 798-3561

CA, NV, OR, WA, HI
Robert Olsen
Nicolet Instrument Corporation
145 East Dana Street
Mountain View, CA 94041
(415) 969-1258

Canada
Allan Crawford Associates
640 - 11th Avenue S. W., Suite 102
Calgary, Alberta T2R 0E2
(413) 261-0780

Allan Crawford Associates
1330 Marie Victorian Boulevard East
Longueuil, P. Q. J4G 1A2
(514) 670-1212
TWX: 610-422-3875

Allan Crawford Associates
6427 Northam Drive
Mississauga, Ontario, L4B 1J5
(416) 678-1500
TWX: 610-492-2119

Allan Crawford Associates
1299 Richmond Road
Ottawa, Ontario K2B 7Y4
(613) 829-9651
TWX: 610-562-1670

Allan Crawford Associates
234 Brooksbank Avenue
North Vancouver, BC V7J 2C1
(604) 980-4831

European Countries
Dr. Peter Langner
Nicolet Instrument GmbH
Goerdeler Strasse 48
D-605 Offenbach am Main
West Germany
0611/852028
Telex: 841/4185411

Japan
Takeda Riken Industry Co., Ltd.
1-32-1, Asahi-cho, Nerima-ku
Tokyo 176, Japan
930-4111
Telex 781/272-2140

Australia
ELMEASCO Instruments Pty. Limited
7 Chard Road
Brookvale, N. S. W. 2100
Australia

New Zealand
ELMEASCO Instruments Pty. Limited
P. O. Box 30515
Lower Hutt
New Zealand