INTRODUCTION TO PROGRAMMING OF THE SX

1. Introductory Exercise

Assume a simple calculation

X = A x B + C

Where A = 10

B = 20

C = 30

To solve this in keyboard mode:-

a) Make sure SX is in OPE mode and that 'Printer Off' & 'Program Select' Buttons are up.

b) Key 10 x 20 + 30 =

c) SX will print log as shown below

```
        10    x
        20    +
        30    =
230.000000
```

Fig. 1

Now let's try programming this:-

The mode in which the SX stores programs is 'LEARN' mode

i) To invoke 'LEARN' mode press the 'LRN' key

ii) Depress C, C-ALL to clear the SX's memory

In 'LEARN' mode the SX will automatically print the step number alongside the instruction given.

- You start off the SPnn where n is a character in the range 0 - 9 or A - F

- 'SP' is short for 'start program' - it denotes the starting point of a program and inserts a special flag so that the start of any program may be easily found.

- SPnn is a double or two-step instruction - after keying the 'SP' a light marked 'UNFIN' will come on to denote that the instruction is unfinished.

- The 'nn' is added to differentiate one program's start point from another's. We will call this one Program 00 (zero zero), so we key SP00

- Then key the keyboard instructions keyed before - this time they will not be executed but will be stored and printed.

- Then depress the 'PRINT' key to incorporate a print instruction (appears as a diamond)
- Finally key EPØØ (short for end of program ØØ)

Now depress 'OPE' to put the SX out of 'LEARN' mode and into 'OPERATE' mode and depress 'C' to reset the machine.

Then depress 'START' to start the program.

By now, you should have a printout like that shown in Figure 2

```
0000 SP
0001 00
0002 1
0003 0
0004 ×
0005 2
0006 0
0007 +
0008 3
0009 0
0010 =
0011 ◇
0012 EP
0013 00
```

```
2.3000000000000 02 ◇
```

Fig. 2

If something is wrong, you can very easily put the SX back into 'LEARN' mode (but don't depress C and C ALL this time or you will 'wipe' your program): position yourself to the right step number as shown on the listing by depressing 'STEP SET' and then keying the 3-digit step number of the step to be rekeyed, then rekey the steps as necessary. (Note: the 'Step back' key takes you back to the previous step)

Notice that the answer produced by the 'PRINT' (Diamond) instruction is in 20 column floating-point format. It always appears this way regardless of the setting of the decimal-point wheel. For this reason, it is only used for simple output.

To produce neater, more intelligible output the following technique is used:-

<u>First Step</u> - Convert the number from the internal floating-point format to fixed point representation by the 'FIX' instruction:-

e.g.	FIXØ	FIX5	FIX9
	02	Ø2	Ø2
	Round Down	Round off	Round Up

- to two decimal places

<u>Second Step</u> - Specify a column print instruction: COL
nn

Where nn is the number of columns to be occupied by the printout including sign and decimal point. Naturally the number of columns specified must be large enough to contain the maximum size of number printed with the number of decimal places shown in the 'FIX' Instruction or an error will result.

So to clean up the print out in this example, put the SX into 'LEARN' mode, Step Set 011 and key the following steps:-

FIX5	(keyed as 'FIXnn', '5')	(Round Off)
02		(to 2 places)
COL		(Column Print)
08		(to 8 places)
EP		
00		

Then revert to 'OPE' mode, depress 'C' and then 'START'.
The printout by now should look like that in Figure 3

```
0011 FIX5
0012 02
0013 COL
0014 08
0015 EP
0016 00



230.00
```

Fig. 3

By now, one monstrous deficiency of the program should be apparent - it will only work for values of A, B & C of 10, 20 & 30: any other values of A, B & C require alteration to the program.

So what should be done is to allow the operator to key in new values of A, B & C each time the program is run.
How is this to be achieved?

2. Data Entry

The SX will stop, turn the 'ENT' light on (and wait for the, operator to key data and press 'START' before resuming) under 2 circumstances:-

a) When it encounters an 'SP' instruction in the program (NB this does not happen when a program is called as a subroutine - explained below)

b) When it encounters an 'ENTRY' instruction - obtained by depressing the 'ENT' key and appearing on the listing as 'E'.

```
0000 SP
0001 00
0002 x
0003 E
0004 +
0005 E
0006 =
0007 FIX5
0008 02
0009 COL
0010 08
0011 EP
0012 00


230.00
```

Fig. 4

In Figure 4, the 'SP00' at the start allows the operator to enter the first number - A. The first ENT allows B to be entered, the second ENT allows C to be entered. The '=' calls out the result from the SX's own working registers into the A register for round-off and print out (every algebraic expression has to terminate in an '=' to recall the result).

If the 'Printer-off' button is left up, the SX will log items entered on the printer; if it is depressed, only explicit print instructions (PRINT, COLUMN - PRINT) will result in print output.

You will notice that after keying the 3 data items, and after printout of the result, the 'ENT' light will again come on. This is because the 'EP00' statement causes the SX to look for the start of program 00, and await input again. Thus there is no reason why a program cannot have several EP's, as the EP merely says 'go back to start! A program can obviously have only one 'SP'

3. Introduction to the User of Memories

What if we wanted to store A, B or C for later use in the program? The answer is simple - the instruction 'SMnn' (nn is a 2 digit memory no. between 00 and 99) says: 'store the number currently in the A - Register into the nominated memory, leaving the A - register untouched'. When we need to recall the contents of the memory for calculation or printout, we issue the instruction 'RMnn' (short for 'Recall Memory') to bring it back from the nominated memory into the A-Register.

'SMnn' and 'RMnn' are, like the 'SPnn' and 'EPnn' instructions, Double instructions. When, for example, keying in the 'SMnn' instructions in 'LEARN' Mode, you depress the 'SMnn' key; the step number and 'SM' will be printed on the listing, and the 'UNFINISHED' light will be displayed - the 2 digit memory number is then keyed.

Try the example shown in Figure 5.

```
0000 SP
0001 00
0002 SM
0003 01
0004 E
0005 SM
0006 02
0007 E
0008 +
0009 RM
0010 01
0011 ×
0012 RM
0013 02
0014 =
0015 FIX5
0016 02
0017 COL
0018 08
0019 LF
0020 LF
0021 EP
0022 00

  230.00
```

Fig. 5

Here it has been decided to retain A & B in memories 1 & 2. For illustration purposes, the calculation is performed using Memories 1 & 2.

So, in Figure 5, the number (A) entered at the 'SP' is stored in Memory

01 (SM01), the next (B) is stored in Ø2, and the next (C) is left sitting in the A-Register as we do not (in this case) wish to retain it after the calculation which follows. RM01 and RM02 recall the contents of memories 1 and 2 as the calculation proceeds.

It is very easy to forget the purposes for which you have assigned memories, so it is advisable to make a list of them as you go. This will also make it very much easier for anyone else reading your program to understand it.

Some other useful memory instructions are:-

CMnn	Clear memory nn (for clearing totals)
ΣMnn	Add the contents of the A-Register to Memory nn, leaving the contents of the A register unchanged.

Notice all data movement and calculation has to take place via the A-Register. To copy the contents of Memory 1 to Memory 2, for instance, the following would be required:-

RM
Ø1
SM
Ø2

Also note that the previous contents of the A-Register would be replaced with a copy of Memory 1.

Also note (in Fig. 5 Steps 19-20)the use of the 'LF' (line feed) instruction. This causes the printer to space a line, and is obtained by depressing 'I/O' and then 'Ø' (zero).

In the examples in Figures 4 and 5, it would be very easy for the operator to lose track of when to enter A, when to enter B, and when to enter C. One ideal way of overcoming this is to print instructions on the printer.
How is this to be accomplished?

4. Incorporating Printer Messages into the Program

Apart from 'prompting' the operator, printer messages are also very useful for making the results easier to understand.

To cause a message to be printed in 'Keyboard' or 'Immediate' mode, proceed as follows:-

- Put the SX in 'OPE' mode

for SX100:-

- Depress 'Character Print'
- then depress 'INTnn' followed by the 2 digit code corresponding to the letter required as per the table below (for digits A to F, you will see the letters marked under, the 2 leftmost columns of keys, e.g. 'ARC' = A, 'SIN' = B etc)

  Repeat for successive characters of the message
- Depress 'Character Print' to terminate the message.

SX100 Character Table

Letter	Code	Letter	Code	Letter	Code
A	41	J	4A	S	53
B	42	K	4B	T	54
C	43	L	4C	U	55
D	44	M	4D	V	56
E	45	N	4E	W	57
F	46	0	4F	X	58
G	47	P	50	Y	59
H	48	Q	51	Z	5A
I	49	R	52	Space	20

For SX300:-

- Depress 'Character Print'
- Type the message, using the keyboard. Note that the alternative, alphabetic, values of the function keys are displayed below them on the case: e.g. 'ARC' gives the letter 'A' etc.
- Depress 'Character Print' to terminate the message.

To do the same thing under program control, insert the message (in 'LEARN' mode) at the appropriate spot in the program, preceded by 'Character Print', and followed by another 'Character Print' to terminate the message.

Note that 'Character Print' appears on the listing as 'CHA'.
Note that the full character set is shown in the SX100 and SX300 manuals.
You will see from Figure 6 that our previous example has now had messages inserted so that the operator is told what to enter next

(ENTER A?, ENTER B?, etc); the answer is preceded by the word 'ANS'.

```
0000 SP
0001 00
0002 FLG
0003 01
0004 LF
0005 LF
0006 CHA
0007 E
0008 N
0009 T
0010 E
0011 R
0012
0013 A
0014 ?
0015 CHA
0016 E
0017 SM
0018 01
0019 FIX5
0020 02
0021 COL
0022 08
0023 LF
0024 CHA
0025 E
0026 N
0027 T
0028 E
0029 R
0030
0031 B
0032 ?
0033 CHA
0034 E
0035 SM
0036 02
0037 FIX5
0038 02
0039 COL
0040 08
0041 LF
0042 CHA
0043 E
0044 N
0045 T
0046 E
0047 R
0048
0049 C
0050 ?
0051 CHA
0052 E
0053 FIX5
0054 02
0055 COL
0056 08
0057 LF
0058 +
0059 RM
0060 01
0061 ×
0062 RM
0063 02
0064 =
0065 ÷
0066 15
0067 CHA
0068 A
0069 N
0070 S
0071
0072 CHA
0073 FIX5
0074 02
0075 COL
0076 08
0077 LF
0078 GT
0079 01
0080 EP
0081 00
```

Fig. 6

The printout from running the program is shown in Figure 7.

```
ENTER A?   14.00
ENTER B?   25.00
ENTER C?    3.60
                ANS   353.60


ENTER A?   18.00
ENTER B?   15.00
ENTER C?   26.33
                ANS   296.33
```

Another thing that has had to be done is to avoid the program stopping at SP00, after the program has processed the output for the first set of values. If it did stop at SP00, after encountering EP00 after the first run through the program (EP00 having 'triggered' a search for SP00), then the 'ENT' light would come on without any preceding printout as to what the operator was supposed to enter.

To avoid this, we have to stop the program reaching EP00, and instead of relying on EP00 to take us back to the start of the program, we instruct the SX (Figure 6, steps 78-79): GT01 - Meaning 'go to Flag 01' - obtained by depressing the 'GO TO nn' key, then keying'01'. We define Flag 01, as being the point before, the printout 'ENTER A?', by keying 'FLAGnn' and 01 at Lines 2-3 (Fig. 6).

Under these circumstances the SP00 and EP00, at the beginning and end of the program respectively, are no longer required, as the GTØ1 and FLGØ1 have effectively taken over their function in this case. They are only left in for clarity's sake.

5. Jumps

Quite often, as in the previous example, the program needs to JUMP to a point other than one which would be reached naturally. These unconditional jumps are achieved by inserting GO TO nn at the point in the program at which the jump is to take place. The 'nn' is a 2 digit code used to define where you want to JUMP to, in combination with a FLAG nn instruction. For consistent results, the 'nn' code associated with the FLAGnn must be unique within the program. The 'GO TO' triggers a search of memory for a FLAG with the same code. 'GO TO' is obtained by depressing the 'GO TO nn' key, followed by the 2 digits of the flag; it appears on the listing as 'GT'. 'FLAG' is obtained by depressing the 'FLAG nn' key, again followed by the 2 digit code. 'FLAG' appears on the listing as "FLG'.

Each digit in the code must be in the range Ø-9, A-F.

So a 'GT55' instruction will trigger a search of memory for a FLG55, if the SX cannot find it, the search will continue over and over again in an endless loop (key 'C' to terminate).

Using this method of jumping, the SX will be searching for a symbol (FLG + Code) - hence this is known as a symbolic jump.

There is another method - an absolute jump. This is implemented by loading the step number (to be jumped to) into the A-Register, by computation, recalling memories, etc. and then issuing the IOF instruction (Keyed as 'I/O', F). This is a much faster instruction because the SX does not have to search - it 'knows' which step to go to. However it is inadvisable to use this approach during program development, as any change to a 'destination' step no. through insertion/deletion of other steps would necessitate changing all the IOF sequences pointing to that step. Consequently, it is usual to insert the IOF's only when the program is virtually in final form.

6. Subroutines

Notice from Figure 6 that certain blocks of steps are repeated several times, viz:-

CHA
E
N
T   Lines 6-11, 24-29, 42-47
E
R

FIX5
Ø2
COL   Lines 19-23, 37-41, 53, 56, 73,77
Ø8
LF

It is tedious to have to enter these steps over and over again, and also consumptive of steps.

To overcome this problem, we can make use of a technique known as subroutining.

Using this technique, we code a frequently - used routine as a program on its own, beginning with 'SPnn' and ending with 'EPnn'.

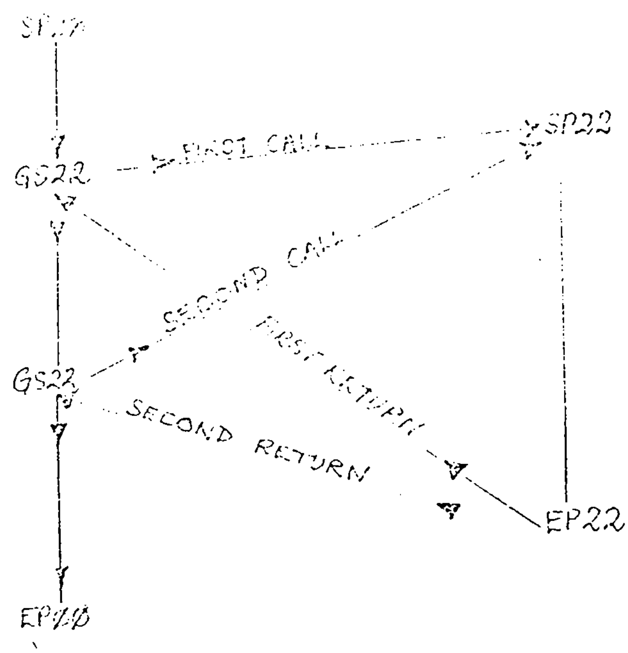
When we want to use this routine, we use the instruction 'GO TO SPnn' (short for 'Go to Subprogram') followed by the 2 digit code assigned to the subprogram. The 'GO To SPnn' key prints as GS.

When a subroutine is called via the 'GS' instruction (eg GS21) the SX searches for an 'SP' with the same code - in this case SP21.

When it finds it, it transfers program control to the new subprogram without stopping for input.

When the subprograms 'EP' is encountered (End of Program), the SX, knowing that the subroutine was called as a subroutine (subprogram), transfers program control back to the step after the 'GS' which called the subroutine. Where there are many calls in the same main program to the same subroutine (via several 'GS's' in various parts of the program), the SX keeps track of which 'GS' called the subroutine, so as to be able to return to the correct section of the program, eg:-

MAINPROGRAM SUB-ROUTINE



Control always transfers back to the step after the 'GS' which called it.

Note the difference between the effect of 'EP' in a program called as a 'main' program and 'EP in a program called as a subprogram:-

Main Program: 'EP' causes search for 'SP'

Subprogram: 'EP' causes return to calling program

The same routine can be used as a main program and as a subprogram - the difference lies in whether it was called via 'GS' or not.

The subroutine can be located anywhere in memory.

```
0000 SP
0001 00
0002 FLG
0003 01
0004 LF
0005 LF
0006 GS
0007 02
0008 CHA
0009 A
0010 ?
0011 CHA
0012 E
0013 SM
0014 01
0015 GS
0016 03
0017 GS
0018 02
0019 CHA
0020 B
0021 ?
0022 CHA
0023 E
0024 SM
0025 02
0026 GS
0027 03
0028 GS
0029 02
0030 CHA
0031 C
0032 ?
0033 CHA
0034 E
0035 GS
0036 03
0037 +
0038 RM
0039 01
0040 ×
0041 RM
0042 02
0043 =
0044 ÷
0045 15
0046 CHA
0047 A
0048 N
0049 S
0050
0051 CHA
0052 GS
0053 03
0054 GT
0055 01
0056 EP
0057 00
0058 SP
0059 02
0060 CHA
0061 E
0062 N
0063 T
0064 E
0065 R
0066
0067 CHA
0068 EP
0069 02
0070 SP
0071 03
0072 FIX5
0073 02
0074 COL
0075 08
0076 LF
0077 EP
0078 03
```

Fig. 9

Figure 9 shows the Program in Figure 6 converted to use two subroutines, one of which (SPØ2) prints 'ENTER' the other (SPØ3) does the rounding off and printing. The 'GS' statements are underlined. The number of steps saved in this case is trivial, but this technique can frequently save a great many steps and greatly simplify programming.

7. Tables & Dissections

Let us say that we want to do a sales analysis whereby we key in the territory number (1 - 50) and then the invoice amount, for all invoices, in such a manner that 50 territory totals are accumulated, for printout after we have finished the last invoice.

Up to now, when using memories we have always specified the memory number to be used. But with 50 possible memory numbers, this would be impossibly tedious.

Luckily, the SX provides a very easy way round the problem whereby, instead of storing in a memory specified in the program, we can get the program to put the number of the memory to be used into a 'pointer' memory (any memory can be used for this purpose). To specify that the register nominated is to be used as a pointer, we precede the memory reference instruction with 'IND' (indirect) obtained by depressing the 'INDIRECT' key.
Contrast these two approaches:-

A	B
	(Memory 20 contains 8)
SM	IND
08	SM
	20

Both achieve the same effect - in case B the SX sees 'IND' and knows that for the following 'Store Memory' instruction it has to treat the contents of memory 20 as the pointer to where it really has to store the data, instead of storing the data directly into Memory 20.

To do our sales analysis, let's use Memories 1 - 50 for the 50 territory totals, and 51 as storage for the territory number. Then, in order to ensure that the invoice total is accumulated in the correct memory, we just designate 51 as a 'pointer' memory.

```
0000 SP
0001 00
0002 f1        CLEAR ALL MEMORIES
0003 FLG
0004 01
0005 LF
0006 CHA
0007 T
0008 E
0009 R
0010 R
0011 I
0012 T
0013 O
0014 R
0015 Y
0016 ?
0017 CHA
0018 E
0019 SM
0020 51        STORE TERRITORY NO IN MEMORY 51
0021 FIX0
0022 00
0023 COL
0024 05
0025 CHA
0026
0027
0028 S
0029 A
0030 L
0031 E
0032 S
0033 ?
0034 CHA
0035 E
0036 FIX5
0037 02
0038 COL
0039 08
0040 IND    ACCUMULATE
0041 SM     IN TERRITORY TOTAL
0042 51
0043 GT
0044 01
0045 EP
0046 00
```

FIG 10

Figure 10 shows a program to do this: the operator is asked for the territory number, and then the sales, which are accumulated in the appropriate territory total in memories 1- 50. Notice the useof the 'F1' instruction at line 2 to clear all memories - this is obtained by keying 'INSTnn', F, 1.

At line 44, the program goes back to line 3 (after the F1 instruction) for the next entry, in anendless loop.

```
TERRITORY?     1  SALES?     1.00
TERRITORY?     2  SALES?     2.00
TERRITORY?     3  SALES?     3.00
TERRITORY?     4  SALES?     4.00
TERRITORY?     1  SALES?     1.00
TERRITORY?     2  SALES?     2.00
TERRITORY?     3  SALES?     3.00
TERRITORY?     4  SALES?     4.00
TERRITORY?
```

```
2     RM01
4     RM02
6     RM03
8     RM04
0     RM05
```

Fig. 11

Figure 11 shows the printout of the program when run('OPE' mode, 'C', & 'START' - 'PRINTER OFF' down).

After the last entry, it was necessary to depress 'C' to stop the program, and put the 'PRINTER OFF' button up, in order to get the printout shown by manually recalling memories 1 - 5 (tedious for 50!)

To recall a memory, you simply key 'RMnn' followed by the memory no:- Ø1, Ø2, Ø3 etc.

Wouldn't it be nice to have this printout occur automatically?
To achieve this, you have to master 2 new skills, - Use of Conditions, and Loops

8. Use of Conditions

It is often necessary for a program to make a logical decision. In the case discussed above, it would be nice to have the machine detect whether the last entry has been keyed.

This might be achieved by use of the 'IF ENT' test. This test checks whether the operator has entered anything in response to the 'ENT' command. It is entered by keying:-

IF GO TO nn
ENTRY
dd

('dd' is the two-digit code of the flag to which the program is to go if something (even a zero) has been keyed)

If appears on the listing as, eg:-

IFE
Ø5

So to terminate the entries, the operator would depress 'START' without keying anything.

Other conditional Tests are available for testing the contents of the A-Register after an arithmetic operation:-

Purpose	Keyed As	Lists As
Is A Non-Zero?	If GO TO, =, dd	IFNZ dd
Is A Positive or Zero	If GO TO, +, dd	IF+ dd
Is A Negative	If GO TO, -, dd	IF- dd

So to test whether the number contained in Memory Ø2 is less than or equal to that in Ø1, and if so to go to FLAG 80, the following could be used:-

RM
Ø1
-
RM
Ø2
= (don't forget to recall result with '=')
IF+
80

If the number in Memory 2 is greater than that in Memory 1, (test is not true), control will 'fall through' the 'IF' and continue at the following step.

Other forms of test are listed in the SX Programmer's Manual.
The instructions to test whether the operator keyed anything are shown in Figure 12, Steps 18-20.

```
0000 SP
0001 00
0002 f1
0003 FLG
0004 01
0005 LF
0006 CHA
0007 T
0008 E
0009 R
0010 R
0011 I
0012 T
0013 O
0014 R
0015 Y
0016 ?
0017 CHA
0018 E
0019 IFE     DID OPERATOR KEY ANYTHING ?
0020 02      YES - GO TO 02
0021 GT
0022 09      NO - GO TO 09
0023 FLG
0024 02
0025 SM      STORE TERRITORY NO
0026 51      IN MEMORY 51
0027 FIX0
0028 00
0029 COL
0030 05
0031 CHA
0032
0033
0034 S
0035 A
0036 L
0037 E
0038 S
0039 ?
0040 CHA
0041 E
0042 FIX5
0043 02
0044 COL
0045 08
0046 IND       ACCUMULATE INVOICE
0047 ΣM        TOTAL IN TERRITORY
0048 51           TOTAL
0049 GT
```

```
0055 LF
0056 LF
0057 LF
0058 CHA
0059 T
0060 E         PRINT
0061 R
0062 R
0063 I         REPORT
0064 T
0065 O
0066 R         HEADINGS
0067 Y
0068
0069
0070 T
0071 O
0072 T
0073 A
0074 L
0075
0076 S
0077 A
0078 L
0079 E
0080 S
0081 CHA
0082 LF
```

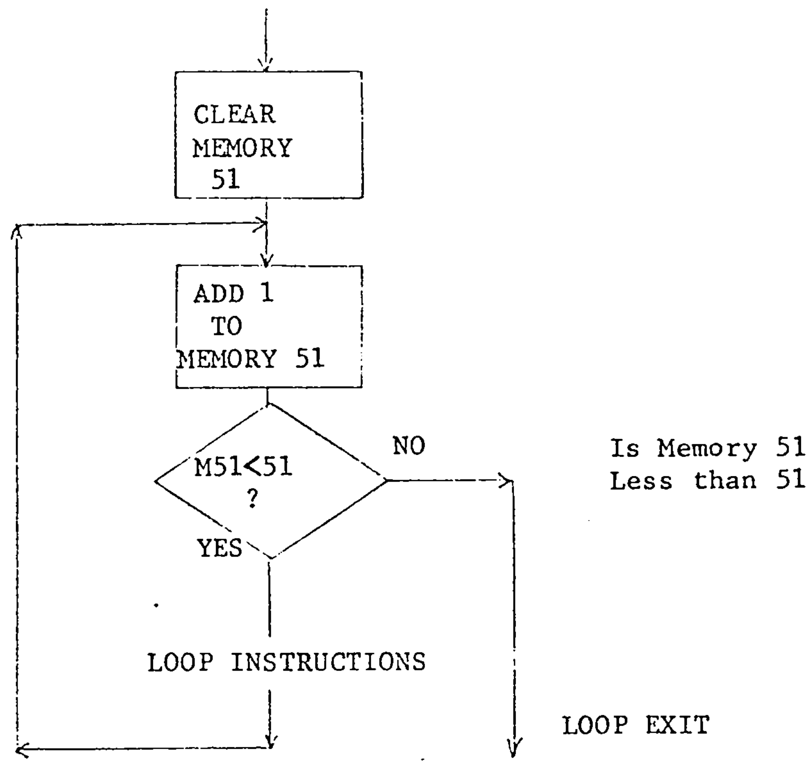
```
0084 FLG
0085 12
0086 1
0087 ΣM    ADD 1 TO LOOPCOUNT
0088 51
0089 RM
0090 51
0091 -
0092 5     IS
0093 1     LOOPCOUNT STILL
0094 =     LESS THAN 51?
0095 IF-
0096 10    YES - CONTINUE
0097 GT    NO - GO TO
0098 99       END
0099 FLG
0100 10
0101 IND   RECALL
0102 RM    TERRITORY TOTAL
0103 51
0104 IFNZ  IS IT NON-ZERO
0105 11    YES-GO PRINT
0106 GT    NO - SKIP THIS
0107 12       LINE
0108 FLG
0109 11
0110 RM    RECALL LOOPCOUNT
0111 51
0112 FIX0
0113 00
0114 COL   PRINT AS
0115 07    TERRITORY NO.
0116 →     SPACE ACROSS
0117 04    4 POSITIONS
0118 IND
0119 RM
0120 51
0121 FIX5  RECALL
0122 02    TERRITORY
0123 COL   TOTAL & PRINT
0124 08
0125 LF
0126 GT    BACK TO START
0127 12    OF LOOP
0128 FLG
0129 99
0130 EP
0131 00
```

L
O
O
P

9. Loops

In the Sales Analysis case under discussion, we want to step through the pointer through the values 1 - 50, so that we can print out the corresponding totals.

To do this we can construct a simple loop, Using Memory 51 as a counter:-



The steps to do this (one way) are shown on lines 83-97 and 126-127 Other refinements added are as follows:-

a) In lines 100-104 we test if the Territory total is Non-Zero - if so, we'll print it, otherwise we'll skip that territory and go on to the next.

b) In lines 115-116 we make use of another handy print formatting instruction - 'SPACEnn' (prints as sideways arrow) which means 'space nn positions' in this case 4 print positions.

The run of the program is shown in Figure 13.

```
TERRITORY?    1   SALES?     1.00
TERRITORY?    2   SALES?     2.00
TERRITORY?    3   SALES?     3.00
TERRITORY?    4   SALES?     4.00
TERRITORY?    1   SALES?     1.00
TERRITORY?    2   SALES?     2.00
TERRITORY?    3   SALES?     3.00
TERRITORY?    4   SALES?     4.00
TERRITORY?   10   SALES?    10.00
TERRITORY?   35   SALES?    35.00
TERRITORY?   50   SALES?    50.00
TERRITORY?

TERRITORY  TOTAL SALES
      1         2.00
      2         4.00
      3         6.00
      4         8.00
     10        10.00
     35        35.00
     50        50.00
```

Fig. 13

10. Function Keys

Frequently the operator needs to be able to call up a special routine such as printing totals (as in the previous example), entering a credit, doing an error correction, etc.

A Function Key facility is provided for this purpose:- when the 'PROGRAM SELECT' button is down, the following 6 keys change their function and, in OPE mode, become function keys:-

ARC	(A)	e^x	(F)
SIN	(B)	a^2	(U)
COS	(C)		
TAN	(D)		
a	(E)		

Depression of a function key causes the SX to look for a routine starting with 'SP' and then a reserved code according to the following:-

Key	Code
A	8A
B	8B
C	8C
D	8D
E	8E
F	8F
U	89

So a routine to be activated by the 'C' key will start:-

SP
8C (etc)

For this to occur, the 'Program Select' button must be down, and the SX must either be idle or in the 'ENT' state (awaiting entry).

A special keyboard overlay is available to remind the operator which function key performs what function.

Another way of calling up a special function when in 'OPE' mode is to depress 'GO TO SPnn' followed by the routine's 2 digit code.

10. Conclusion

Hopefully, this has served as a primer to the use of the SX; it is intended to be read in conjunction with the SX Programming Manuals

In particular, the following topics are not covered in this Introduction:-

- Use of Check Mode for Inserting and Deleting steps
- Use of debug Mode for Debugging programs
- Scientific functions
- Use of Magnetic Card & Cartridge
- Splitting Memories
- Full SX Instruction Set

For these consult the following Canon Publications:-

SX Programming manual

SX Programmable Calculator Instructions

SX Scientific Functions Instructions