

**IBM 650 COMPUTER  
STONE & WEBSTER ASSEMBLY PROGRAM**

**SWAP MANUAL**



**STONE & WEBSTER  
ENGINEERING CORPORATION**



100173

STONE & WEBSTER ENGINEERING CORPORATION

I.B.M. 650 COMPUTER  
STONE & WEBSTER ASSEMBLY PROGRAM

SWAP MANUAL  
BY  
ALEXANDER ZABLUDOWSKY

JUNE 1, 1957



## FOREWORD

The Stone & Webster Assembly and Interpretive Programs (SWAP and SWIP) described in this manual were written primarily in order to enable more engineers, presently unfamiliar with computers, to learn in as short a time as possible the art of programming.

With this idea in mind, a single address, sequential interpretive routine was devised, employing programmed index registers. This system, having few instructions, yet retaining flexibility, should be within easy grasp of any engineer. Since it operates in floating point, it should be well suited for most engineering problems. The interpretive program was designed also to facilitate program debugging. Yet, in order to make it as easy to grasp as possible, very few restrictions were placed in its use, and the ones found necessary will cause an easily recognizable programmed alarm. The routine is "foolproof", outside of direct intervention. Psuedo operation codes were chosen so that most of them will be illegal as machine codes.

The assembly program was designed to permit the use of completely symbolic programs, with the utmost flexibility in the use of symbols. It permits a semiautomatic use of the interpretive, and other often used, routines. The availability of parameters and relative addresses should be helpful in the use of subroutines which are frequently needed in engineering



work. Here again, utmost flexibility is allowed and the fewest number of restrictions imposed, yet each error will be easily detected by means of a programmed alarm. The phonetic codes were chosen for their mnemonic value and easy spotting. Punching of symbol tables and availability cards should prove to be of great help in debugging.

The system was designed to permit the use of the assembly and interpretive routines as a package, without concern for the basic machine code which is harder for a novice to understand. As the engineer, through use of the interpretive programs, becomes familiar with the concept of digital computation, the step to machine code and a fuller use of the system should prove to be easy. This is a distinct advantage over available three address systems.

An acknowledgment is due to the Digital Computer Laboratory of M.I.T. and to I.B.M., from whose programs many concepts and even phonetic codes were borrowed.

June 1, 1957

A. Zabłudowsky

## TABLE OF CONTENTS

|                                                 | <u>Pages</u> |
|-------------------------------------------------|--------------|
| INTRODUCTION                                    | 1-5          |
| CHAPTER 1 THE SWAP COMPUTER, SIMPLIFIED VERSION | 6-15         |
| CHAPTER 2 INDEX REGISTERS                       | 16-21        |
| CHAPTER 3 SOME MORE FUNDAMENTALS                | 22-32        |
| CHAPTER 4 SUBROUTINES                           | 33-38        |
| CHAPTER 5 SWAP FORMAT                           | 39-47        |
| CHAPTER 6 SWAP OUTPUT AND CORRECTING TECHNIQUES | 48-52        |
| CHAPTER 7 ADVANCED PROGRAMING TECHNIQUES        | 53-56        |
| CHAPTER 8 ADVANCED SWAP OPERATION AND FORMAT    | 57-66        |
| APPENDIX AVAILABLE STORAGE CHART                |              |
| SWAP INSTRUCTION CHART                          |              |
| THE SWIP PROGRAM                                |              |
| THE IPC AND IRD PROGRAMS                        |              |
| THE READ-IN ROUTINE                             |              |
| THE SWAP PROGRAM                                |              |
| SWAP BLOCK DIAGRAM                              |              |
| SWAP OPERATING INSTRUCTIONS                     |              |
| I.B.M. 650 MANUAL                               |              |



## INTRODUCTION

A digital computer is an electronic device which is capable of performing arithmetic operations on numerical data. To perform these arithmetic operations, the computer contains an arithmetic element which is designed to do addition, subtraction, multiplication or division. These operations are performed much in the same way as in a desk calculator, provided the data and an instruction denoting the type of operation desired, are made available. The arithmetic element performs these operations extremely fast, but its efficiency depends primarily on the speed with which data and instructions are made available to it. Had the data and instructions been fed by hand, the device would be only slightly more efficient than a desk calculator and its high cost would render it completely impractical. It was, therefore, necessary to create a device which would speed the transfer of data and instructions to the arithmetic element so that it could perform operations at its maximum speed, with minimum time spent in waiting for information. The device which makes this possible is the memory or storage element.

The memory is a device in which both the data and the instructions can be stored and from which they can be made available to the arithmetic element at a high rate of speed. Thus, a program composed of the required data and instructions is first stored in the memory. Then the computations are carried out by the arithmetic element, drawing its necessary

information from storage. In order to distinguish between the data and the instructions, a control element is included. This element investigates the nature of the instruction in storage and sets the arithmetic element to perform it. The control element also controls the sequence by which the instructions are performed.

In order to read information into and out of memory, an in-out element is included.

Memory - The memory or storage element in the I.B.M. 650 computer is a magnetic drum on which information can be recorded in much the same fashion as on magnetic tape. The drum is divided into 2000 storage registers (much like pigeon holes). In each of these, a word (or number or an instruction) can be stored. Each register is numbered so that the information stored in it can be referred to. The number of the register is referred to as its address. Thus, if it is desired to refer to a specific word, (be it a number or an instruction) stored in a specific location, it can be accomplished by specifying the address of this location.

Each word contains ten decimal digits and a sign. Data will be stored as ten digit signed numbers. Instructions will be stored so that two digits denote the type of operation desired and the rest denote the address of the data on which this operation is to be performed.



Arithmetic Element - The arithmetic element contains an accumulator in which a ten digit number can be stored. An operation is performed between this number and another number stored in the memory, and the result of the operation again stored in the accumulator. An instruction, therefore, will specify not only the desired operation, but also the address of the register in which the second number is stored.

The computer, therefore, is similar to a desk calculator installation. In fact, it is the same aside from being automatic.

In a desk calculator installation, we would have an in-out basket (in-out element) to which problems will be brought and from which results can be taken. The problems would be stated in some code which the desk calculator operator (control element) can understand. He will then, with the aid of a notebook (memory), make the necessary computations on the desk calculator (arithmetic element). The computer as described above would be quite inconvenient to use since it understands only numerically coded information and, like a desk calculator, requires a careful watch over the position of the decimal point. To ease this problem, it is possible to create programs (sets of instructions) that will "teach" the computer a different, easier to remember code and enable it to keep track by itself of decimal points, etc. Such programs are the Stone & Webster assembly program (SWAP) and Stone & Webster interpretive program (SWIP).

To clarify this point, we can compare the computer to a monkey. This bright animal can do many tricks, but to make it perform them, one would have to know monkey language, provided such a language exists. To make the monkey more useful, we would have to teach it a new language simpler than the language of human beings, since a monkey is a simple soul, but still close to human language so that a human could learn it easily. Next we would teach it the desired tricks so that it could show a profit. The language here corresponds to SWAP which is a system that converts an easier language to computer language. The tricks are included in SWIP which enable the computer to perform more desired operations and keep track of the decimal point at the same time. Thus, we can use the monkey without worrying that it get out of hand and, by using a simpler language, make it easier to tell the monkey what to do and make it possible for the monkey to tell us what ails it without our being familiar with monkey anatomy and psychology. In order to have this convenience, a certain amount of storage has to be taken up by these "teaching" routines, thus not all 2000 registers are made available to the programmer. (For exact memory capacity, see chart in Appendix.) This system, in effect, saves program preparation or programming time at the expense of solution time since the newly "taught" instructions are performed by means of previously programmed sequences of machine instructions.



It can therefore be seen that in order to achieve a solution to a given problem on the computer, two steps are necessary, as follows:

1. Planning the program or sequence of elementary steps by which the problem may be solved.
2. Coding the sequence of steps into a coded program - a sequence of instructions the computer can understand.

CHAPTER ITHE SWAP COMPUTER  
SIMPLIFIED VERSION

Introduction - The SWAP computer, as described in the introduction, includes the basic computer elements: an arithmetic element with an accumulator, a control element, a memory consisting of approximately 1740 storage registers, and an in-out element.

Numbers used in the computer occupy a single register and are composed of eight significant digits that range in magnitude between  $10^{-50} \leq n < 10^{49}$ .

A number smaller than  $10^{-50}$  will be taken as a zero and a number larger than  $10^{49}$  will cause a machine alarm and a stop. In the accumulator, a number can be generated in the range  $10^{-(10^{10})} \leq n < 10^{(10^{10})}$ , but a number exceeding  $10^{49}$  could not be stored in the memory.

An instruction occupies one storage register and signifies both the type of operation desired and the address in storage from which the data are to be taken. Instructions are normally performed sequentially so that after the instruction located in A1 (where A1 stands for an arbitrary address  $0 \leq A1 < 1740$ ) the instruction in  $A1 + 1$  will be performed.



Definition of Terms -

|            |                                                 |
|------------|-------------------------------------------------|
| MRA        | - Accumulator (SWIP)                            |
| C(MRA)     | - The contents of the accumulator (SWIP)        |
| C'(MRA)    | - New contents of accumulator after computation |
| A1         | - Address of a chosen register                  |
| C(A1)      | - The contents of register A1                   |
| —————→     | - Replaces                                      |
| Reset (A1) | - Set contents of register A1 to +0             |

Transfer Instructions - In order to perform a computation, it is necessary to transfer data in and out of the accumulator. To start a computation, a number has to be brought into the accumulator and when the computation is completed, the result has to be stored back in memory. For this purpose, the following instructions are available:

|        |                                 |                                                                                                                            |
|--------|---------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| IRA A1 | Reset and<br>add                | Reset the accumulator and add to<br>it the contents of register A1<br><br>$C(A1) \longrightarrow C(MRA)$                   |
| IRS A1 | Reset and<br>subtract           | Reset the accumulator and subtract<br>from it the contents of register A1<br><br>$-C(A1) \longrightarrow C(MRA)$           |
| IRM A1 | Reset and<br>add mag-<br>nitude | Reset the accumulator and add to<br>it the absolute value of the<br>contents of A1<br><br>$ C(A1)  \longrightarrow C(MRA)$ |

In the above instructions, the contents of register A1 are not disturbed by the transfer.

|        |          |                                                                           |
|--------|----------|---------------------------------------------------------------------------|
| IEX A1 | Exchange | Exchange the contents of the accumulator with the contents of register A1 |
|        |          | $C(A1) \longrightarrow C(MRA) \quad C(MRA) \longrightarrow C(A1)$         |
| IST A1 | Store    | Store the contents of the accumulator in register A1                      |

The store instruction does not disturb the contents of the accumulator. Thus, if we had three numbers located in registers A1, A1+1, A1+2, which we wanted to store in registers A1+1, A1+2 and A1+3, respectively, the program might read:

|    |          |    |                |
|----|----------|----|----------------|
| B1 | IRA A1   | A1 | n <sub>1</sub> |
|    | IEX A1+1 |    | n <sub>2</sub> |
|    | IEX A1+2 |    | n <sub>3</sub> |
|    | IST A1+3 |    |                |

After the transfer is completed, the contents of A1, A1+1, A1+2 and A1+3 would be:

|    |                |
|----|----------------|
| A1 | n <sub>1</sub> |
|    | n <sub>1</sub> |
|    | n <sub>2</sub> |
|    | n <sub>3</sub> |

Where B1 is the location of the first instruction of the program and n<sub>1</sub>, n<sub>2</sub>, n<sub>3</sub> are the numbers.

If we now desired to transfer the absolute value of the three numbers from locations A1+1, A1+2, A1+3 to A2, A2+1, A2+3, respectively, the program would read:



9.

|    |          |    |                |
|----|----------|----|----------------|
| B2 | IRM A1+1 | A1 | n <sub>1</sub> |
|    | IST A2   |    | n <sub>1</sub> |
|    | IRM A1+2 |    | n <sub>2</sub> |
|    | IST A2+1 |    | n <sub>3</sub> |
|    | IRM A1+3 | A2 | a <sub>1</sub> |
|    | IST A2+2 |    | a <sub>2</sub> |
|    |          |    | a <sub>3</sub> |

At the completion of the program, the contents of A2, A2+1, A2+2 would be:

|    |                |
|----|----------------|
| A2 | n <sub>1</sub> |
|    | n <sub>2</sub> |
|    | n <sub>3</sub> |

where a<sub>1</sub>, a<sub>2</sub> and a<sub>3</sub>, which were the contents of these registers previous to the transfer, are erased. On the other hand, we would have n<sub>3</sub> stored in the accumulator at the end of the transfer since the transfer instruction did not disturb it.

If we now desired to store the negative of n<sub>1</sub> in A3, the program would read:

|    |        |    |                |
|----|--------|----|----------------|
| B3 | IRS A1 | A1 | n <sub>1</sub> |
|    | IST A3 |    | n <sub>1</sub> |
|    |        |    | n <sub>2</sub> |
|    |        |    | n <sub>3</sub> |
|    |        | A3 | a <sub>4</sub> |

At the end of this transfer -n<sub>1</sub> will be stored in the accumulator and register A3, with register A1 undisturbed, and a<sub>4</sub>, previously in register A3, erased.

Arithmetic Instructions - The most important instructions from the engineering point of view are the ones denoting arithmetic operations. The following are made available:

|     |    |     |                                                                    |
|-----|----|-----|--------------------------------------------------------------------|
| IAD | A1 | Add | Add to the contents of the accumulator the contents of register A1 |
|-----|----|-----|--------------------------------------------------------------------|

$$C'(MRA) = C(MRA) + C(A1)$$

|     |    |          |                                                                           |
|-----|----|----------|---------------------------------------------------------------------------|
| ISU | A1 | Subtract | Subtract from the contents of the accumulator the contents of register A1 |
|-----|----|----------|---------------------------------------------------------------------------|

$$C'(MRA) = C(MRA) - C(A1)$$

|     |    |               |                                                                                          |
|-----|----|---------------|------------------------------------------------------------------------------------------|
| IAM | A1 | Add magnitude | Add to the contents of the accumulator the absolute value of the contents of register A1 |
|-----|----|---------------|------------------------------------------------------------------------------------------|

$$C'(MRA) = C(MRA) + C(A1)$$

|     |    |                    |                                                                                                 |
|-----|----|--------------------|-------------------------------------------------------------------------------------------------|
| ISM | A1 | Subtract magnitude | Subtract from the contents of the accumulator the absolute value of the contents of register A1 |
|-----|----|--------------------|-------------------------------------------------------------------------------------------------|

$$C'(MRA) = C(MRA) - C(A1)$$

|     |    |        |                                                                       |
|-----|----|--------|-----------------------------------------------------------------------|
| IDV | A1 | Divide | Divide the contents of the accumulator by the contents of register A1 |
|-----|----|--------|-----------------------------------------------------------------------|

$$C'(MRA) = \frac{C(MRA)}{C(A1)}$$

|     |    |                    |                                                                         |
|-----|----|--------------------|-------------------------------------------------------------------------|
| IMR | A1 | Multiply and round | Multiply the contents of the accumulator by the contents of register A1 |
|-----|----|--------------------|-------------------------------------------------------------------------|

$$C'(MRA) = C(MRA) \times C(A1)$$

In all the above operations, 8-digit accuracy is preserved.

If we wanted to evaluate  $y$  given in the expression

$$y = \frac{a_1 x}{a_3} - \frac{a_2 x^2}{a_3} + |a_4| x^3$$

where  $x$  is stored in register X1,  $a_1$ ,  $a_2$ ,  $a_3$  and  $a_4$  are stored in registers A1, A1+1, A1+2 and A1+3, respectively, and we wanted  $y$  stored in register Y1, the program might read



11.

|    |          |    |    |
|----|----------|----|----|
| B1 | IRA A1+1 | X1 | x  |
|    | IMR X1   | Y1 | o  |
|    | IMR X1   | A1 | a1 |
|    | IDV A1+2 |    | a2 |
|    | IST T1   |    | a3 |
|    | IRA A1   |    | a4 |
|    | IMR X1   |    |    |
|    | IDV A1+2 | T1 | o  |
|    | ISU T1   |    |    |
|    | IST T1   |    |    |
|    | IMR A1+3 |    |    |
|    | IMR X1   |    |    |
|    | IMR X1   |    |    |
|    | IMR X1   |    |    |
|    | IAD T1   |    |    |
|    | IST Y1   |    |    |

where the program starts in register B1 and register T1 is used for temporary storage. Similarly, the same expression for y could be written:

$$y = \left[ \left( \left( a_4 \mid a_3x - a_2 \right) x + a_1 \right) \frac{x}{a_3} \right]$$

and the program would read:

|    |          |    |    |
|----|----------|----|----|
| B1 | IRM A1+3 | X1 | x  |
|    | IMR A1+2 | Y1 | o  |
|    | IMR X1   | A1 | a1 |
|    | ISU A1+1 |    | a2 |
|    | IMR X1   |    | a3 |
|    | IAD A1   |    | a4 |
|    | IMR X1   |    |    |
|    | IDV A1+2 |    |    |
|    | IST Y1   |    |    |

Here the number of registers used for the program was appreciably cut and no temporary register was necessary. Thus it can be seen that many ways exist for programming the same computation. Usually the way which is most obvious and rapid as far as programming is concerned consumes more storage and is slower as far as machine time is concerned.

Jump Instructions - In programming, it is often desired to break the sequence of instructions. If this were not possible, programs could run only in a straight line and have a maximum of 1740 instructions to perform, beginning with the instruction located in address 0000 and ending in the instruction located in address 1739. This would be a serious handicap, since it is often desired to repeat a certain part of the computation a number of times. If the instructions themselves were repeated the required number of times, the amount of storage available might not be sufficient. To permit a break in sequence, the following instructions are available:

|     |    |                  |                                                                                                                                                                                                                                    |
|-----|----|------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IJU | Al | Jump             | Take the next instruction from location Al and continue from there.                                                                                                                                                                |
| IJN | Al | Jump if negative | Take the next instruction from location Al and continue from there provided the contents of the accumulator are negative. If the contents of the accumulator are positive, ignore this instruction and continue in sequence.       |
| IJP | Al | Jump if positive | Take the next instruction from location Al and continue from there, provided that the contents of the accumulator are positive. If the contents of the accumulator are negative, ignore this instruction and continue in sequence. |



|     |    |                 |                                                                                                                                                                                                                                                                                                                       |
|-----|----|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| IJZ | A1 | Jump if<br>zero | Take the next instruction from<br>location A1 and continue from<br>there provided the absolute<br>value of the contents of the<br>accumulator is smaller than $10^{-50}$ .<br>If the absolute value of the<br>accumulator is greater or equal<br>to $10^{-50}$ , ignore this instruction<br>and continue in sequence. |
|-----|----|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

The above instructions do not disturb the contents of the accumulator. In these instructions, the address A1 does not refer to the location of data but to the location of an instruction. Thus, if we wanted to evaluate a function  $z = a_3y$

|                  |           |
|------------------|-----------|
| $y = a_1x + a_2$ | $x < a_3$ |
| $y = a_4x + a_5$ | $x = a_3$ |
| $y = a_6x + a_7$ | $x > a_3$ |

Where  $x$  is stored in register X1,  $z$  is to be stored in register Z1 and  $a_1, a_2, a_3, a_4, a_5, a_6$  and  $a_7$  are stored in A1, A2, A3, A4, A5, A6 and A7, respectively. The program would read

|    |     |    |    |       |
|----|-----|----|----|-------|
| B1 | IRA | X1 | X1 | x     |
|    | ISU | A3 | Z1 | 0     |
|    | IJN | B2 | A1 | $a_1$ |
|    | IJZ | B3 | A2 | $a_2$ |
| B5 | IRA | X1 | A3 | $a_3$ |
|    | IMR | A6 | A4 | $a_4$ |
|    | IAD | A7 | A5 | $a_5$ |
|    | IJU | B4 | A6 | $a_6$ |
| B2 | IRA | X1 | A7 | $a_7$ |
|    | IMR | A1 |    |       |
|    | IAD | A2 |    |       |
|    | IJU | B4 |    |       |
| B3 | IRA | X1 |    |       |
|    | IMR | A4 |    |       |
|    | IAD | A5 |    |       |
| B4 | IMR | A3 |    |       |
|    | IST | Z1 |    |       |

Here the program starts at B1. The program for  $x < a_3$  starts at B2, for  $x = a_3$  at B3, and for  $x > a_3$  at B5. At B4, we evaluate  $z$  and store it in Z1.

In the machine, both a +0 and a -0 can be stored. The result of a subtraction or addition would always lead to a +0. A zero, unless otherwise specified, is represented by  $0x10^{-50}$  or smaller and only this kind of a zero will cause a jump by the IJZ instruction.

As a further example, if it was desired to raise  $x$  to the 50th power, having  $x$  stored in register X1, it would be possible to do so by first resetting and adding IRA X1 and then repeating the instruction IMR X1, 49 times. This, of course, would be wasteful of storage. Therefore, a program could be written that would repeat the IMR X1 instruction 49 times without actually having 49 of these instructions. This could be done as follows:

|    |     |    |    |       |
|----|-----|----|----|-------|
| B1 | IRA | X1 | X1 | $x$   |
|    | IST | T1 | T1 | 0     |
| B2 | IRA | T1 | A1 | -49.0 |
|    | IMR | X1 | A2 | +1.0  |
|    | IST | T1 |    |       |
|    | IRA | A1 |    |       |
|    | IAD | A2 |    |       |
|    | IST | A1 |    |       |
|    | IJN | B2 |    |       |
|    | IRA | T1 |    |       |



15.

Here  $x$  is first stored in T1, which is used as temporary storage. The number of multiplications is counted by adding 1 to the -49.0 stored in A1. When 49 multiplications have been completed, A1 would contain a +0. At the end of the program, the accumulator will contain  $x^{50}$ .

CHAPTER 2  
INDEX REGISTERS

Very often, in writing a program, the need for repeating an almost identical set of instructions several times might arise. Here the jump instructions described in the previous chapter will be of no help since they eliminate only the need for repeating identical sets of instructions. As an illustration, if it was desired to evaluate the scalar product of two vectors A and B,

$$A.B = a_1b_1 + a_2b_2 + a_3b_3 + \dots + a_nb_n$$

the program might read as follows:

|    |     |            |  |    |                |
|----|-----|------------|--|----|----------------|
| C1 | IRA | A1         |  | A1 | a <sub>1</sub> |
|    | IMR | B1         |  |    | a <sub>2</sub> |
|    | IST | T1         |  |    | a <sub>3</sub> |
|    | IRA | A1 + 1     |  |    | .              |
|    | IMR | B1 + 1     |  |    | .              |
|    | IAD | T1         |  |    | .              |
|    | IST | T1         |  |    | a <sub>n</sub> |
|    | IRA | A1 + 2     |  | B1 | b <sub>1</sub> |
|    | IMR | B1 + 2     |  |    | b <sub>2</sub> |
|    | IAD | T1         |  |    | b <sub>3</sub> |
|    | IST | T1         |  |    | .              |
|    | .   |            |  |    | .              |
|    | .   |            |  |    | .              |
|    | .   |            |  |    | b <sub>n</sub> |
|    | .   |            |  | T1 | 0              |
|    | IRA | A1 + (n-1) |  |    |                |
|    | IMR | B1 + (n-1) |  |    |                |
|    | IAD | T1         |  |    |                |
|    | IST | T1         |  |    |                |

If we assume the contents of T1 to be zero at the start of the program, the first three instructions could be changed to:

```
IRA  A1
IMR  B1
IAD  T1
IST  T1
```



and it can be easily seen that this program is made out of  $n$  blocks each of the form

```

      IRA  A1 + i
      IMR  B1 + i
      IAD  T1
      IST  T1

```

where  $i$  ranges from 0 to  $n-1$ .

In order to eliminate the need of repeating this block  $n$  times, three index registers have been included in SWIP. These are numbered 1 through 3. These index registers are designed to permit effective modification in addresses of instructions. Thus, each of the instructions mentioned in the previous chapter can, if it is so desired, indicate an index register (Example: IRA A1 1 refers to index 1). Each index register contains two parts, index and criterion, the criterion being the measure of how high we want the index to get. An instruction which has an index specified will operate on the location specified by the address plus the contents of that index. Thus, if index 3 contains the number 100, the instruction IRA B1 3 will be executed as if it were IRA B1 + 100. In other words, it will reset and add to the accumulator the contents of register B1 + 100. This is called the effective address. If we now change the contents of index 3 to 101 and return to the same instruction, it will then be performed as if it were IRA B1 + 101. Thus, the program for the scalar product of two vectors could now read as follows:

- C1 - Set criterion of index register 1 to n and the index to 0
- C2 - IRA A1 1  
IMR B1 1  
IAD T1  
IST T1
- C3 - Increase index 1 by 1
- C4 - Check if index 1 equals criterion of index 1.  
If it does, continue to next instruction;  
if not, return to C2.

This program, partly written in statements instead of instructions (tagged by C1, C3 and C4), will perform the desired scalar vector multiplication. The instructions that perform the equivalent of these statements are:

|     |    |   |               |                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-----|----|---|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ICR | A1 | n | Set criterion | Set the criterion of index n ( $1 \leq n \leq 3$ ) to the value A1; set index n to zero.                                                                                                                                                                                                                                                                                                                                                                    |
| ICT | A1 | n | m Count       | Increase the contents of index n ( $1 \leq n \leq 3$ ) by the quantity m. Compare the new contents of this index to the contents of the criterion. If the contents of the index are smaller than the contents of the criterion, take the next instruction from A1 and continue from there. If the contents of the index are equal to or greater than the contents of the criterion, set the index to zero and continue to the next instruction in sequence. |

It can be seen that the ICR instruction performs exactly the statement tagged by C1 and the ICT instruction performs the two statements tagged by C3 and C4. It can also be seen that the address of the ICR instruction does not refer to a location in storage but has a definite numeric value. This value can range in magnitude from 0 to 9999. In the ICT



instruction we introduced a new value designation by m. This value is called increment and it designates the size of step desired each time the calculation is repeated. Thus, the example given at the beginning of the chapter will read:

|    |     |    |     |    |       |
|----|-----|----|-----|----|-------|
| C1 | ICR | n  | 1   | T1 | 0     |
| C2 | IRA | A1 | 1   | A1 | $a_1$ |
|    | IMR | B1 | 1   |    | $a_2$ |
|    | IAD | T1 |     |    | .     |
|    | IST | T1 |     |    | .     |
|    | ICT | C2 | 1 1 |    | .     |
|    |     |    |     |    | $a_n$ |
|    |     |    |     | B1 | $b_1$ |
|    |     |    |     |    | $b_2$ |
|    |     |    |     |    | .     |
|    |     |    |     |    | .     |
|    |     |    |     |    | $a_n$ |

While if a and b were stored so that they alternate in memory, the program would read:

|    |     |    |     |    |       |
|----|-----|----|-----|----|-------|
| C1 | ICR | 2n | 1   | T1 | 0     |
| C2 | IRA | A1 | 1   | A1 | $a_1$ |
|    | IMR | B1 | 1   | B1 | $b_1$ |
|    | IAD | T1 |     |    | $a_2$ |
|    | IST | T1 |     |    | $b_2$ |
| C3 | ICT | C2 | 1 2 |    | .     |
|    |     |    |     |    | .     |
|    |     |    |     |    | .     |
|    |     |    |     |    | $a_n$ |
|    |     |    |     |    | $b_n$ |

The section of the program between C2 and C3 is called a loop and this loop will be repeated by the computer n times. Since there are three separate index registers, it is possible to program three loops, one inside the other. Once

the ICR instruction is performed, the criterion of the specified index will remain unchanged until a second ICR instruction specifying the same index is performed. Another two index instructions are made available to the programmer:

ICI A1 n Increase count Add the value A1 to the contents of index n ( $0 \leq n \leq 3$ ) and proceed to the next instruction.

ICD A1 n Decrease count Subtract the value A1 from the contents of index n and proceed to the next instruction.

These instructions enable the programmer to design loops with irregular count. As with the ICR instruction, the address A1 does not refer to a location, but has a definite numeric value. All four index instructions described in this chapter must have an index register specified. All other instructions described previously may have no index designation, and in this case, they will be performed in the normal fashion described in the previous chapter. Thus the last example of the previous chapter, in which it was desired to raise x stored in register X1 to the 50th power and store the result in T1 would now read:

|    |     |    |     |    |   |
|----|-----|----|-----|----|---|
| A1 | ICR | 49 | 1   | X1 | x |
|    | IRA | X1 |     | T1 | 0 |
| A2 | IMR | X1 |     |    |   |
|    | ICT | A2 | 1 1 |    |   |
|    | IST | T1 |     |    |   |



As a further example, if we wanted to form  
 $t = a_1b_1 + a_2b_2 + \dots + a_8b_8$  where  $a_1 a_2 \dots a_8$  and  $b_1 b_2$   
 $\dots b_8$  are stored in groups of two with each group separated  
 from the other by four registers, the program would read:

|    |     |                |   |   |    |                |    |                |
|----|-----|----------------|---|---|----|----------------|----|----------------|
| C1 | ICR | 2 <sup>4</sup> | 1 |   | T1 | 0              | B1 | b <sub>1</sub> |
| C2 | ICR | 2              | 2 |   | A1 | a <sub>1</sub> |    | b <sub>2</sub> |
| C3 | IRA | A1             | 1 |   |    | a <sub>2</sub> |    | 0              |
|    | IMR | B1             | 1 |   |    | 0              |    | 0              |
|    | IAD | T1             |   |   |    | 0              |    | 0              |
|    | IST | T1             |   |   |    | 0              |    | 0              |
|    | ICI | 1              | 1 |   |    | 0              |    | b <sub>3</sub> |
|    | ICT | C3             | 2 | 1 |    | a <sub>3</sub> |    | b <sub>4</sub> |
|    | ICT | C2             | 1 | 4 |    | a <sub>4</sub> |    | 0              |
|    |     |                |   |   |    | 0              |    | 0              |
|    |     |                |   |   |    | 0              |    | 0              |
|    |     |                |   |   |    | 0              |    | 0              |
|    |     |                |   |   |    | 0              |    | 0              |
|    |     |                |   |   |    | .              |    | .              |
|    |     |                |   |   |    | .              |    | .              |
|    |     |                |   |   |    | .              |    | b <sub>7</sub> |
|    |     |                |   |   |    | .              |    | b <sub>8</sub> |
|    |     |                |   |   |    | a <sub>7</sub> |    |                |
|    |     |                |   |   |    | a <sub>8</sub> |    |                |

CHAPTER 3SOME MORE FUNDAMENTALS

As mentioned in the Introduction, a 650 storage register is made up of 10 decimal digits and a sign. However, all the instructions covered in the previous chapter included alphabetic information which is meaningless to the machine. The conversion of this alphabetic information to numerical code is one of the functions of SWAP. By first running a program containing alphabetic information through the SWAP routine a numerically coded program can be obtained. The final numerically coded program will be performed by the interpretive program (SWIP). The numeric code used for representing numbers or instructions is meaningful only to the SWIP program and not to the basic 650 computer. In fact, most of the numerically coded information would be illegal to the basic 650. Thus, the need arises for a designating instruction which will tell the machine that the program following it is a SWIP program. This instruction is:

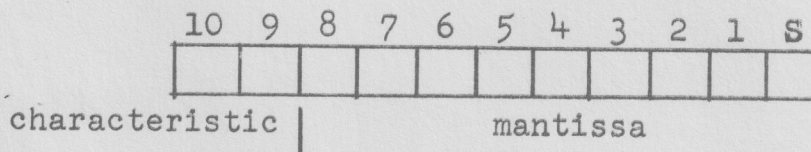
IN                    into SWIP                    perform the following instructions in the SWIP mode.

This instruction has no address section and should be immediately followed by the desired program. Only one IN instruction is necessary and it should be located at the very start of the program.



Numbers - SWIP numbers are of the form  $n = a \times 10^b$ .

Where  $a$  is called the mantissa and  $b$  the exponent. The mantissa is always a normalized fraction; that is, its magnitude is in the range  $0.1 \leq a < 1.0$ . Since fractions are used throughout the machine, the largest possible number of significant digits is preserved. The mantissa of a SWIP number is composed of eight decimal digits and a sign. For example, the number  $+56.78$  will be represented as  $+0.56780000 \times 10^2$  and the number  $-0.00375$  as  $-0.37500000 \times 10^{-2}$ . Since only one sign is available in a register, the exponent is modified by adding 50 to it; the new value is then called the characteristic. Thus, the allowable exponent is in the range  $-50 \leq b \leq 49$  and the characteristic will then be positive and in the range 0 to 99. The eight digit signed mantissa and the two digit characteristic occupy one register in the form



where  $s$  signifies the sign (for more description of the 650 register, see 650 manual). The digits 1-10 designate the ten decimal digits. Thus the number

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S |
| 5  | 4 | 6 | 7 | 8 | 9 | 3 | 2 | 1 | 0 | + |

is the machine equivalent for  $+6789.321$

and the number

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S |
| 4  | 8 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | - |

is the machine equivalent for  $-0.0012345678$

The Accumulator - When a number of the form described above is transferred to the accumulator, it is unpackaged. The accumulator occupies two signed registers, one for the mantissa and one for the exponent. Thus, the characteristic is transformed back into an exponent to facilitate the performance of arithmetic operations. If we have in register 100 the number  $-357.1234$  which is in the form

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S |
| 5  | 3 | 3 | 5 | 7 | 1 | 2 | 3 | 4 | 0 | - |

and the instruction IRA 100 is performed, the following will be in the accumulator:

|          |   |   |   |   |   |   |   |   |   |   |          |   |   |   |   |   |   |   |   |   |   |
|----------|---|---|---|---|---|---|---|---|---|---|----------|---|---|---|---|---|---|---|---|---|---|
| 10       | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S | 10       | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S |
| 3        | 5 | 7 | 1 | 2 | 3 | 4 | 0 | 0 | 0 | - | 0        | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 3 | + |
| Mantissa |   |   |   |   |   |   |   |   |   |   | Exponent |   |   |   |   |   |   |   |   |   |   |

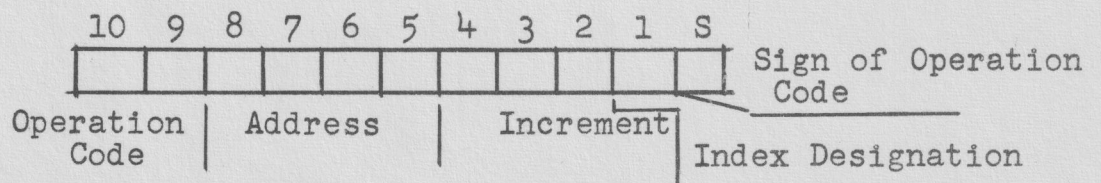
During arithmetic operations on this number, 10 digit accuracy will be preserved and the exponent will be allowed to range from  $-10^{10}$  to  $+10^{10}$ . The magnitude of the mantissa still is within the limit of from 0.1 to 0.9999999999. When the instruction IST or IEX is performed, the number in the accumulator is packaged again to its original form and stored in the specified location. Several errors can be encountered as a result of computations. The first one is a result of the exponent exceeding the limit of  $-10^{10}$  to  $+10^{10}$ . In such a case, the first significant digit of the exponent will be lost, thus resulting in error. A machine alarm will occur which will stop the machine with the



console overflow light on. A second error can be encountered if the instruction IST or IEX is performed on a number in the accumulator the exponent of which exceeds +49. Here the characteristic will be larger than +99 and its first significant digits lost. In this case, a programmed alarm will occur and the machine will stop with the number 5000 showing on the address lights of the console. When an exponent -50 or smaller is encountered during an IST or IEX instruction, the number will be taken as a 0 and the characteristic resulting will be 0, with no alarm occurring. A third error might result from the mantissa exceeding +0.9999999999. Here again, the first significant digits will be lost. This condition can arise only if a division by an unnormalized number or by zero is attempted, and will result in a machine stop with the overflow console light on.

All arithmetic operations will end with a normalized result in the accumulator. Therefore, the overflow resulting from division by or of an unnormalized number could not be achieved in normal operation.

Instructions - All instructions mentioned in the previous chapters were composed of four sections: Operation Code, Address, Index Designation and Increment. These sections are converted by SWAP into an instruction occupying a single register in the form



Thus, the instruction IRA 305 1 will be converted to

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S |
| 0  | 0 | 0 | 3 | 0 | 5 | 0 | 0 | 0 | 1 | + |

Where +00 is the operation code for IRA. All instructions can have a nonzero increment part, but it will be ignored in all but the ICT and ITA (see next chapter) instructions. If no index is specified, a 0 will be placed in the index designation digit. (For list of the operation codes, see end of chapter.) An instruction looks much the same as a number in storage except that most instructions are equivalents of unnormalized numbers. The only way for the computer to distinguish between numbers and instruction is by the sequence of the instructions. Great care must be used therefore, in writing a program so that the sequence will not lead the computer to a number which is a meaningless instruction. The following will cause a programmed alarm and a machine stop: (1) An illegal operation code will stop the computer with the number 3000 displayed on the console address light. (2) An illegal index designation, meaning either that the digit valve is greater than 3 or that no index is specified in an ICR, ICT, ICI or ICD instruction, will stop the computer with the number 4000 or 6000, respectively, displayed on the console address lights.

Addresses - Three types of addresses can be used in the program. The first of these is an absolute address which is of the form 1234 or 77. It can have anywhere from 1 to 4



digits and in most cases designates a specific location. In the ICR, ICI and ICD instructions, such an address would designate a specific quantity which is used with the index registers. Since the 650 storage includes 2000 registers addressed from 0 to 1999, an instruction referring to an address larger than 1999 would be illegal and will stop the machine, with the illegal address displayed on the console address lights. This is not the case with the ICI, ICR and ICD instructions where the address can go as high as 9999, since it does not refer to a storage location. (Actually, addresses larger than 1740 will disturb the SWIP program and cause a program error.) (Addresses of 8000, 8001, 8002 and 8003 in some cases may not cause an alarm, but will create a program error.) The illegal address is a handy tool by which the programmer can ascertain whether the program has been run successfully. The programmer might write IJU 3333 as the last instruction of the program and when the computer stops and displays 3333 on the console address light, it indicates the program has stopped where desired.

Using absolute addresses, the last example of the previous chapter could be written as follows:

|     |     |      |   |   |
|-----|-----|------|---|---|
| 100 | IN  |      |   |   |
|     | ICR | 24   | 1 |   |
|     | ICR | 2    | 2 |   |
|     | IRA | 500  | 1 |   |
|     | IMR | 550  | 1 |   |
|     | IAD | 499  |   |   |
|     | IST | 499  |   |   |
|     | ICI | 1    | 1 |   |
|     | ICT | 103  | 2 | 1 |
|     | ICT | 102  | 1 | 4 |
|     | IJU | 3333 |   |   |
| 499 |     | 0    |   |   |
| 500 | a1  |      |   |   |
|     | a2  |      |   |   |
|     | 0   |      |   |   |
|     | 0   |      |   |   |
|     | 0   |      |   |   |
|     | 0   |      |   |   |
|     | a3  |      |   |   |
|     | a4  |      |   |   |
|     | .   |      |   |   |
|     | .   |      |   |   |
|     | .   |      |   |   |
|     | .   |      |   |   |
|     | a7  |      |   |   |
|     | a8  |      |   |   |
| 550 | b1  |      |   |   |
|     | b2  |      |   |   |
|     | 0   |      |   |   |
|     | 0   |      |   |   |
|     | 0   |      |   |   |
|     | 0   |      |   |   |
|     | .   |      |   |   |
|     | .   |      |   |   |
|     | .   |      |   |   |
|     | b7  |      |   |   |
|     | b8  |      |   |   |

The addresses to the left of the instruction code are called tags and they assign the instruction to the specified location, so the IN will be assigned to location 100, the ICR 24 1 to location 101, etc., until a new tag is reached, in this case 499. The computer will then skip all locations in between and



assign the zero to location 499. This assignment is done by the SWAP program in such a way that information can be fed in any order so long as the untagged instructions go in order. In the above example, if the order were changed and the data placed before the instructions, the resulting program would still be the same provided the same tags were used.

The second type of address the programmer can use is the symbolic address. This address is composed of one letter and one or two digits in the form A1, X21 or B99. If a given symbol appears in the address section of some instruction (excluding ICR, ICI and ICD) the same symbol has to appear elsewhere as a tag for some other register. In case of ICR, ICI or ICD the symbol will have to be defined by other means described in Chapter 5. Symbolic addresses can also be of the form A1+5 (a symbol plus or minus a number of up to three digits). A symbol plus or minus another symbol is also allowed. Thus, all the examples given in the previous chapters were those of symbolic programs. These would then be converted by SWAP to absolute addresses. Obviously, duplicate tags can not be used otherwise SWAP will not be able to determine which one is correct. A limit of 250 different symbols is placed by the SWAP conversion program (for more detail, see Chapter 5).

The third type of address that can be used is a relative address of the form 5R, OR, 1000R. This address is composed of one to four digits followed by the letter R. Thus, a relative address nR where n is any number  $0 \leq n \leq 1999$  refers to the nth register after the last tag appearing in the program. (Relative addresses can not be used if no tags preceded them.)

As an illustration of the three types of addresses we shall take the example of raising  $x$  to the 50th power:

|     |     |      |   |   |     |      |   |   |  |     |      |   |   |  |
|-----|-----|------|---|---|-----|------|---|---|--|-----|------|---|---|--|
| 100 | IN  |      |   |   | 100 | IN   |   |   |  | 100 | IN   |   |   |  |
| A1  | ICR | 49   | 1 |   | ICR | 49   | 1 |   |  | ICR | 49   | 1 |   |  |
|     | IRA | X1   |   |   | IRA | 107  |   |   |  | IRA | 7R   |   |   |  |
| A2  | IMR | X1   |   |   | IMR | 107  |   |   |  | IMR | 7R   |   |   |  |
|     | ICT | A2   | 1 | 1 | ICT | 103  | 1 | 1 |  | ICT | 3R   | 1 | 1 |  |
|     | IST | T1   |   |   | IST | 108  |   |   |  | IST | 8R   |   |   |  |
|     | IJU | 3333 |   |   | IJU | 3333 |   |   |  | IJU | 3333 |   |   |  |
| X1  | x   |      |   |   | x   |      |   |   |  | x   |      |   |   |  |
| T1  | 0   |      |   |   | 0   |      |   |   |  | 0   |      |   |   |  |

Symbolic

Absolute

Relative

The three forms shown above or any combination of them would be converted by SWAP to exactly the same numerically coded program. In the symbolic program the first tag (100) could be left out, then, the IN instruction would be located by SWAP in register 0, etc. Thus, we see that symbolic notation permits writing a program without considering storage allocation. Also, program debugging becomes much easier. The relative address form is especially helpful in subroutines (see Chapter 4).

Index Registers - The index registers, as mentioned in the previous chapter, occupy two registers each. One of these registers is occupied by the criterion and the other by the index. In both of these registers only four digits are made available, thus, great care should be exercised so that the index does not exceed 9999 or become negative. Index designation has to be done by a digit, no symbols are permissible.

Increment - As already mentioned, all instructions can specify an increment but in all instructions beside ICT and



ITA (see next chapter) it will be ignored. The increment can be specified by a symbol of one letter and one or two digits but in such a case the symbol has to be previously defined (see Chapter 5).

Alarms - The following is a list of possible alarms that can occur during run of a SWIP program:

| <u>Type of Alarm</u>   | <u>Description of Stop</u>        | <u>Cause of Alarm</u>                                                                                                                                                           |
|------------------------|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1) Overflow            | Overflow light on                 | 1) Absolute value of exponent greater than $10^{10}$ .<br>2) Division by zero.<br>3) Division by an unnormalized number.<br>4) Contents of index negative or greater than 9999. |
| 2) Overflow            | 5000 on address lights            | Exponent larger than 49 during an IST or IEX instruction.                                                                                                                       |
| 3) Illegal address     | Illegal address on address lights | 1) Illegal effective address in an instruction specifying an index register.<br>2) Illegal address.                                                                             |
| 4) Illegal instruction | 3000 on address lights            | Attempt to execute an instruction with an illegal operation code.                                                                                                               |
| 5) Illegal index       | 4000 on address lights            | Illegal index specified (digit 1 of instruction greater than 3).                                                                                                                |
|                        | 6000 on address lights            | No index designation in ICR, ICT, ICI or ICD instruction.                                                                                                                       |

Timing - The following is the list of times spent in performing each of the instructions. Time, listed is in milliseconds and is the time corresponding to the worst conditions.

| <u>Numeric<br/>Code</u> | <u>Instruction</u> | <u>Name</u>             | <u>Time</u> |
|-------------------------|--------------------|-------------------------|-------------|
| +00                     | IRA                | Reset and add           | 43.0        |
| +01                     | IRS                | Reset and subtract      | 42.0        |
| +02                     | IRM                | Reset and add magnitude | 42.0        |
| +03                     | IEX                | Exchange                | 62.0        |
| +04                     | IST                | Store                   | 42.0        |
| +05                     | IAD                | Add                     | 76.0        |
| +06                     | ISU                | Subtract                | 76.0        |
| +07                     | IAM                | Add magnitude           | 76.0        |
| +08                     | ISM                | Subtract magnitude      | 76.0        |
| +09                     | IDV                | Divide                  | 66.0        |
| +10                     | IMR                | Multiply and round      | 62.0        |
| +11                     | ITA*               | Transfer address        | 50.0        |
| +12                     | IJU                | Jump (unconditional)    | 28.0        |
| +13                     | IJN                | Jump if negative        | 38.0        |
| +14                     | IJP                | Jump if positive        | 38.0        |
| +15                     | IJZ                | Jump if zero            | 35.0        |
| +16                     | OUT**              | Out of SWIP             | 28.0        |
| -17                     | ICR                | Set criterion           | 66.0        |
| -18                     | ICT                | Count                   | 71.0        |
| -19                     | ICI                | Count increase          | 70.0        |
| -20                     | ICD                | Count decrease          | 70.0        |
| Special                 | IN                 | Into SWIP               | 9.6         |

\*See Chapter 4

\*\*See Chapter 7



CHAPTER 4  
SUBROUTINES

Very often in a program the necessity arises for performing the same computation on several sets of data. As an example, if it was desired to evaluate the quantity:

$$y = \cos x + a_1 \cos^2 a_2 x$$

The program might read:

```

B1      IN
        IRA  X1
        Compute cosine
        IST  Y1
        IRA  X1
        IMR  A2
        Compute cosine
        IST  T1
        IMR  T1
        IMR  A1
        IAD  Y1
        IST  Y1
        IJU  4444
X1      x
A1      a1
A2      a2
Y1      o
T1      o

```

Here the two sections described by "compute cosine" would be identical. Both would be sets of instructions that compute the cosine of the quantity stored in the accumulator and leave the result in the accumulator. These sections are called subroutines. Since they are repeated as often as used,

they are called "open" subroutines. We could also write the same program in the following way:

```

B1  IN
    IRA  X1
B2  IJU  C1
    IST  Y1
    IRA  X1
    IMR  A2
B3  IJU  C1
    IST  T1
    IMR  T1
    IMR  A1
    IAD  Y1
    IST  Y1
    IJU  4444
X1  x
A1  a1
A2  a2
Y1  o
T1  o
C1  Compute cosine
C2  Return to the instruction following the IJU C1
    used to enter the subroutine (exit).

```

Here the subroutine is placed only once in the program resulting in a substantial saving in space. The statement tagged by C2 could be satisfied by the instruction IJU with an address that changes each time the subroutine is entered so as to re-enter the main program at the proper point. In order to describe the instruction that performs this function, we shall first further describe the jump instructions.

The jump instructions available and described in the previous chapters are IJU, IJN, IJP, IJZ and ICT. When any one of these instructions is performed in a condition that results in a jump, its own address is placed in a special register inside



SWIP. Thus, if in location 100 we have the instruction IJU A1, when this instruction is performed, the quantity 100 will be placed in the special register. The instruction that will now modify the exit of a subroutine is:

|        |                  |                                                                                                                                             |
|--------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|
| ITA A1 | Transfer address | Take the contents of the special SWIP register, add the increment section to it and place the result in the address section of register A1. |
|--------|------------------|---------------------------------------------------------------------------------------------------------------------------------------------|

This instruction does not disturb the contents of the accumulator.

The contents of the special register are disturbed by every instruction aside from ITA; thus, the ITA instruction, when desired, has to immediately follow the jump instruction. The previous example will now read:

|    |                |      |   |   |
|----|----------------|------|---|---|
| B1 | IN             |      |   |   |
|    | IRA            | X1   |   |   |
| B2 | IJU            | C1   |   |   |
|    | IST            | Y1   |   |   |
|    | IRA            | X1   |   |   |
|    | IMR            | A2   |   |   |
| B3 | IJU            | C1   |   |   |
|    | IST            | T1   |   |   |
|    | IMR            | T1   |   |   |
|    | IMR            | A1   |   |   |
|    | IAD            | Y1   |   |   |
|    | IST            | Y1   |   |   |
|    | IJU            | 4444 |   |   |
| X1 | x              |      |   |   |
| A1 | a <sub>1</sub> |      |   |   |
| A2 | a <sub>2</sub> |      |   |   |
| Y1 | o              |      |   |   |
| T1 | o              |      |   |   |
| C1 | ITA            | C2   | 0 | 1 |
|    | Compute cosine |      |   |   |
| C2 | IJU            | 0    |   |   |

Here the IJU at C2 will read IJU B2 + 1 after the first entry, and IJU B3 + 1 after the second entry. The subroutine from C1 to and including C2 is called a "closed" subroutine. Thus, with this subroutine in the program an IJU C1 is equivalent to computing the cosine of the contents of the accumulator.

Many of the often used subroutines such as cosine, sine, tangent, square root, etc., will be available at the computer subroutine library so that the programmer will be saved the trouble of programming them. These subroutines will be written in relative addresses so that no restrictions are imposed on the programmer's choice of symbols. The programmer would then tag them by a symbol or absolute address of his choice. Thus, the example given previously will now read:

|    |                       |      |
|----|-----------------------|------|
| B1 | IN                    |      |
|    | IRA                   | X1   |
| B2 | IJU                   | C1   |
|    | IST                   | Y1   |
|    | IRA                   | X1   |
|    | IMR                   | A2   |
| B3 | IJU                   | C1   |
|    | IST                   | T1   |
|    | IMR                   | T1   |
|    | IMR                   | A1   |
|    | IAD                   | Y1   |
|    | IST                   | Y1   |
|    | IJU                   | 4444 |
| X1 | x                     |      |
| A1 | a <sub>1</sub>        |      |
| A2 | a <sub>2</sub>        |      |
| Y1 | o                     |      |
| T1 | o                     |      |
| C1 | LSR cosine subroutine |      |



Where the operation code LSR is a pseudo instruction which stands for "library subroutine" and is used to tag the subroutine with the desired symbol, (for a description of pseudo codes see Chapter 5) a full description of the use of each of the library subroutines will be available.

Two subroutines are made available to the programmer on a semiautomatic basis, by this we mean that when their use is indicated by the programmer a reservation will be automatically made for them in storage. Thus, the use of these two subroutines will decrease the number of available storage registers. These two subroutines are designated by the two instructions:

|     |       |                                                                                             |
|-----|-------|---------------------------------------------------------------------------------------------|
| IPC | punch | Punch the contents of the accumulator in a number card format. (See Chapter 5.)             |
| IRD | read  | Read one numeric card and place its converted contents in the accumulator. (See Chapter 5.) |

Both instructions have no address section. In order for the programmer to use either of these instructions, an indication to that effect is necessary at the start of the program. (See Chapter 5.) Thus, if it was desired to evaluate the quantity  $y_i = ax_i + b$   $0 \leq i \leq 20$  where the  $x_i$ s are on cards which are ready in the card reader and the  $y_i$ s are to be punched after each computation, the program would read:

```

C1  IN
    ICT  20  1
C2  IRD
    IMR  A1
    IAD  B1
    IPC
    ICT  C2  1  1
    IJU  5555
A1  a
B1  b

```

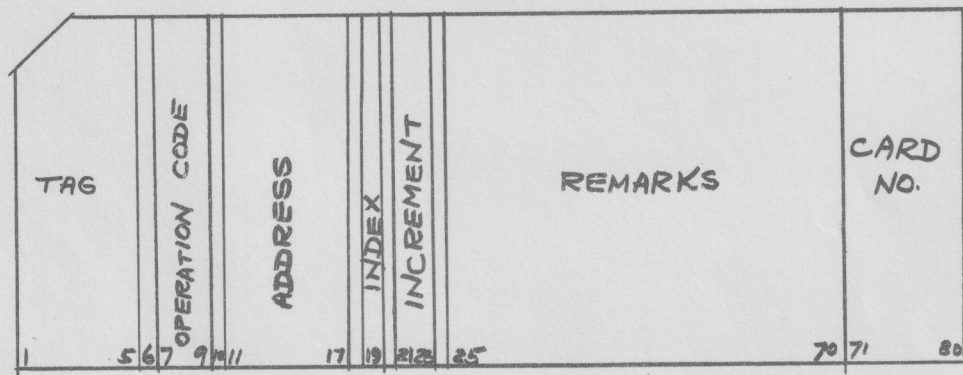
If the exponent of the number punched is greater than 49, same alarm as in IST will occur. If the number read in is of the wrong form the computer will stop on a validity alarm.



CHAPTER 5SWAP FORMAT

The input and output from the computer are by means of punched cards. These cards consist of 80 columns each of which can be punched to represent either a decimal digit or a letter. (For further description of the card format, see I.B.M. 650 manual.) SWAP reads these cards and converts them to numerically coded cards. There are three basic types of cards designed to accommodate the three types of input desired.

Instruction Cards - Instruction cards are of the following form:



The tag can be either empty, symbolic, absolute, relative or parameter (see below). If no tag or a previously undefined symbolic tag is specified, the instructions will be allocated to the register immediately following the location assigned to the last word. If the tag is a previously defined symbol, an absolute or a relative address, the appropriate location will be assigned. A parameter tag will place the numerically converted

instruction in the parameter table, here all symbols used must be previously defined. A symbolic or parameter tag must start from the left while an absolute or relative tag must end to the right. Thus the symbol or parameter letter should be punched in column 1 and the least significant digit or the letter R of an absolute or relative tag should be punched in column 5. The operation code must be one of the recognized codes covered in the preceding chapters. If the codes IPC or IRD are used, indication to that effect must be made on the HED card (see below).

The address section can be symbolic, absolute or relative. A symbolic address has to start from the left and if it has an addition of the form  $A21 + 5$  or  $B99-C1$ , the sign must appear in column 14 immediately followed by the addition. In computing the address absolute values will be used, thus, if  $B99-C1$  leads to a negative number, an error would result. Absolute or relative addresses should end to the right so that the least significant digit or the letter R occupy column 17. If an address is punched in either IN, IRD or IPC instructions, it will be ignored. If no address is specified a zero will be assumed.

The index, if specified, should be punched as a digit in column 19. The increment can be either symbolic or absolute.

Remarks and card number may be punched from column 25 on. These will be ignored by SWAP.



Number Cards - To read a number into the computer, the following card format is used:

| TAG | FLO   | MANTISSA | ±  | EX<br>PO<br>N<br>E<br>N<br>T | REMARKS | CARD<br>NO. |
|-----|-------|----------|----|------------------------------|---------|-------------|
| 1   | 5 7 9 | 11       | 22 | 24 25                        | 27      | 70 71 80    |

Here the tag will be of the same form as in the instruction cards.

The operation code FLO will be punched in columns 7-9. This indicates to SWAP that the card contains a number.

The mantissa should be punched to as many digits as desired in columns 11 through 22 with its sign in column 22. If a parameter symbol is punched in the mantissa field the appropriate entry in the parameter table will be assigned to the location described by the tag. If no entry appears a zero will be assumed. The mantissa is normalized and rounded to eight significant digits by SWAP. The decimal point is assumed to be just before the first digit. The exponent should be punched in columns 24-25 with its sign in column 25. Thus the number +345.7890157 will be punched as +3457890157 +3 or as

+003457890157 +5. In both cases it will be converted to 5334578902+ by SWAP. Again remarks and card number may be

punched in columns 27-80. Cards which are to be read by IRD must be in normalized form with first digit of the mantissa in column 11 and all eight digits punched. The exponent should have both digits punched.

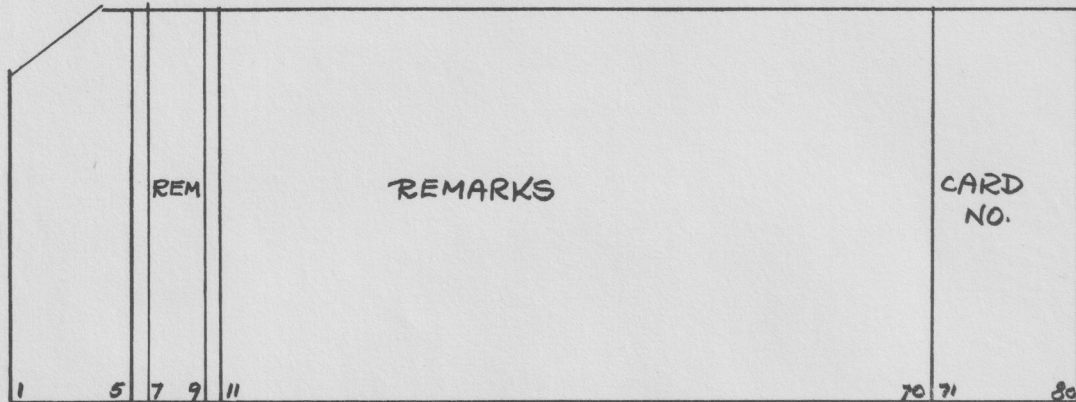
Pseudo Instruction Cards - Pseudo instructions are instructions to the assembly program as to the method of conversion desired. They are not converted into numerically coded information and do not occupy storage locations. The Pseudo instructions available are:

HED - The HED card is the first card of the program. It designates whether the following program uses SWIP, IPC or IRD (IRD and IPC can be used only when SWIP is designated). This card will also include the problem number, deck number and programmer's number. This information will be repunched on the final numerically coded deck for identification purposes. The card format is as follows:

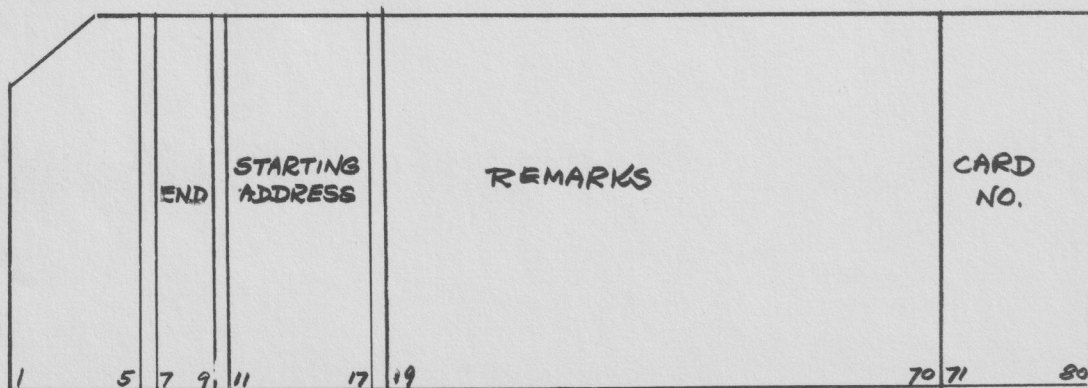
| SWIP ?<br>IPC ?<br>RD ?<br>AVAILABILITY? |   |   |   | H<br>U | PROBLEM NO. | DECK NO. | PROGRAMMER'S NO | TITLE AND REMARKS | CARD<br>NO |    |    |    |    |    |    |
|------------------------------------------|---|---|---|--------|-------------|----------|-----------------|-------------------|------------|----|----|----|----|----|----|
| 1                                        | 2 | 3 | 4 | 7      | 9           | 11       | 13              | 15                | 17         | 19 | 20 | 22 | 70 | 71 | 80 |



REM - The REM card is used where the space for remarks on the other card formats is insufficient. Any number of these cards may be used by the programmer and they are ignored by SWAP. The format is:



END - The END card is the last card of the program deck and it designates to SWAP that no more program cards follow. The address section of this card may be symbolic, absolute or relative and will designate to SWAP where the program being converted is to start when it is run. The converted value of this address will be punched on the final card of the numerically coded deck. If the available read-in routine is used, (see below) this card will cause the program to start automatically after reading of the numerically coded deck. If the available read-in routine is not used this card should be discarded.



LSR - The LSR card precedes a library subroutine and must have a tag designation. It assigns this tag to the following library subroutine. This card also resets the parameter table:

|     |          |                  |          |
|-----|----------|------------------|----------|
| TAG | LSR      | NAME AND REMARKS | CARD NO. |
| 1   | 5 7 9 11 |                  | 70 71 80 |

EQU - The EQU card is used to designate a value to a symbol, thus A1 EQU 51 will assign to the symbol A1 the value 51. The same will result from 51 EQU A1. The address section of this card can be symbolic with or without addition, absolute or relative. The tag can be symbolic absolute or relative, but either tag or address must be previously defined and should not cause an inequality. Thus, if A1 was already assigned the value 50, a card A1 EQU 75 would result in an alarm:

|     |          |         |         |          |
|-----|----------|---------|---------|----------|
| TAG | EQU      | ADDRESS | REMARKS | CARD NO. |
| 1   | 5 7 9 11 | 17 19   |         | 70 71 80 |



DIT - The DIT card is used to repeat the preceeding word of the program a desired number of times. Thus, the preceeding number or instruction will be repeated between its location and the address of the DIT card, inclusive. The address of the DIT card may be symbolic if previously defined, absolute or relative, in which case it will repeat the last work up to the specified address relative to the preceeding word. Thus the program

```
100 FLO +532 +5
    DIT 105
```

will result in the following numeric program:

```
100 5553200000 +
101 5553200000 +
102 5553200000 +
103 5553200000 +
104 5553200000 +
105 5553200000 +
```

|   |   |   |   |    | DIT |    | ADDRESS |  | REMARKS  |    |
|---|---|---|---|----|-----|----|---------|--|----------|----|
|   |   |   |   |    |     |    |         |  | CARD NO. |    |
| 1 | 5 | 7 | 9 | 11 | 17  | 19 |         |  | 70       | 71 |
|   |   |   |   |    |     |    |         |  |          | 80 |

Parameters - A parameter is a symbol represented by one letter. It facilitates the change of certain parts of the program from run to run without removing any card from the

deck. The limit on the number of parameters is 26 (the number of letters in the alphabet), but since the LSR card resets the parameter table, a larger number may be used.

Operation - After the program is prepared, it is punched in the format described above and read, together with the SWAP program into the computer. SWAP then converts the program and punches it on cards in numerically coded form. The first card of the program deck must be a HED card and this card will be reproduced for identification on the final deck. The last card of the program deck must be an END card and its converted form will be punched as the last card of the final deck. A read-in program is then available in storage to read the converted final deck. This program is located in registers 0 through 27. Thus, if the programmer desires to use this read-in routine he would have to specify to SWAP not to assign any words into these locations. This can be done by tagging the first word of the program with the absolute address 28 or with a previously equated symbol of that value. When this read-in routine is used, the final deck may be immediately read-in and the computer will start operating on the program automatically. If this read-in routine is not used the last card of the final deck should be discarded and other means of reading and starting used.

When SWAP concludes the conversion, it will punch the symbol table which will include every symbol used in the program



and its assigned equivalent. An appropriate SWIP deck must precede the final deck in reading into the computer before the run is made.

SWAP Alarms - Various alarms may occur during the conversion of the program deck. These are listed in the appendix.

Program Correcting - If errors are found in the program they may be corrected by correction cards which are merely the corrected word in its form tagged with the location into which it should go. These cards may be placed at the end of the program deck just before the END card. If a symbolic tag is used in a correction card, that tag would have been used at least twice within the same program and an indication to this effect will be made in the symbol table.

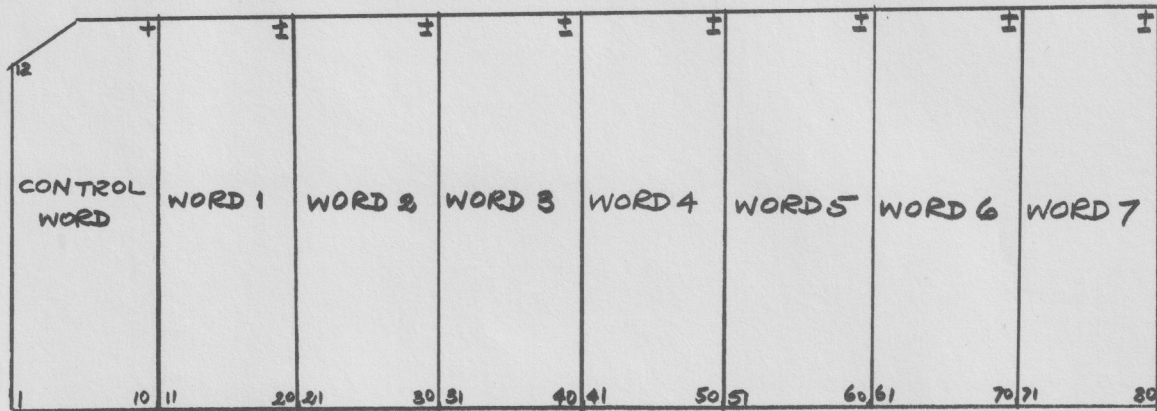
CHAPTER 6SWAP OUTPUT AND CORRECTING TECHNIQUES

The instructions and format covered in the preceding chapters were those employing the SWIP interpretive routine. As was already described, SWIP is a routine written in basic 650 code which interprets numerically coded information to perform a set of instructions. The SWAP assembly program is capable of converting an alphabetic program into that set of numerically coded instructions and data. Thus, running a problem on the computer consists of, first, converting the deck to a numeric code and, second, operating on the numeric code to perform the desired computation. Each of these is a separate step and takes up a certain amount of computer time depending on the length of the program and the number of instructions that have to be performed. Thus, it is not necessary to convert a deck before each run. Once an alphabetic deck has been converted, it can be run any number of times and any corrections necessary can be made to it. This will result in an appreciable savings in machine time. In certain cases, where the number of corrections required is large, it might be advantageous to correct and reconvert the alphabetic deck by the means described previously.

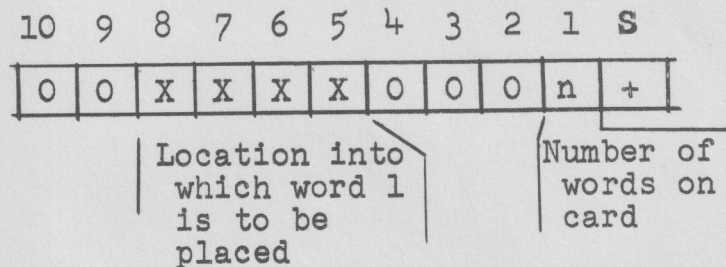
In order to correct the numerical deck, we must first study its format. Since the storage register of the 650 is 10 digits long and the card has 80 columns, it is possible to



punch eight words on a card. Here the sign is punched in the same column as the least significant digit (see 650 manual) and a 12-Punch is made in column 1 to make the card a load card (see 650 manual).



The first word on the card is a control word which signifies where the words following are to be placed in storage and how many of them there are.



Thus, as shown above, n words will be placed in n consecutive locations starting with the specified address. With the aid of the symbol table which is punched at the end of conversion, it is possible to locate the error in the numeric card and correct it by replacement.

The symbol table is punched in alphabetic and numeric order, eight symbols and their assignments on a card. This enables the programmer to locate a word from the symbolic deck in the numerical deck.

Corrections - Two basic types of errors are common in programming, the first of which is a wrong word, be it a number or an instruction. As an example, consider the program:

|    |     |    |     |
|----|-----|----|-----|
| D2 | ICR | 6  | 1   |
| D1 | IRA | A1 | 1   |
|    | IMR | B1 | 1   |
|    | IST | C1 |     |
|    | ICT | D1 | 1 1 |

Assume that the first instruction should read ICR 7 1 instead. Here the correction is easy to make. The word ICR 6 1 may be located by looking up D2 in the symbol table and reproducing the appropriate numeric card so that the word -1700060001 will be changed to -1700070001 without disturbing anything else on the card. By this method converting the alphabetic deck can be avoided. It is also possible to either replace the erroneous card by a corrected card in the alphabetic deck or to place a properly tagged correction card at the end of the alphabetic deck just before the END card. Either of these methods require reconverting the alphabetic deck, thus increasing the machine time spent on the problem.

The second type of error consists of missing numbers or instructions. This can occur either when an instruction or a number has been left out or when an incorrect section of the



program is to be replaced by a new section which has more instructions, thus requiring relocations of part of the program. This condition can be corrected in several ways. If the program was written in symbolic form throughout, it is possible to replace the section directly in the alphabetic deck and reconvert. On the other hand, a method called "patching" may be employed. This method will be the only one possible in a program using absolute addresses. As an example if in the program

```

A1  IRA  B1
    IMR  C1
    IDV  C2
    IST  B2

```

it was found that an instruction IAD C3 has been left out between the IMR and IDV instructions. If the whole program was written in the same symbolic form, it would be possible to insert the missing instruction and reconvert. SWAP will then give new value to some of the symbols and thus create a space for the missing instruction. On the other hand, if the program was in the form

```

100 IRA 201
    IMR 352
    IDV 353
    IST 202

```

it would be impossible to insert the missing instruction unless all absolute addresses were changed to conform. Here a patch can be made by placing an IJU instruction in place of the IMR by the method described previously. The IJU, if properly

addressed will cause the program to proceed to the location just after the last program word. Here we shall write:

|     |     |
|-----|-----|
| IMR | 352 |
| IAD | 354 |
| IJU | 102 |

which is now the desired program (IAD 354 was the missing instruction). The patch takes up two more locations than the number of missing instructions. The same method could have been used on the symbolic program and is the only possible method if it is desired to correct the final numeric deck.

When any SWIP alarm occurs, register 1827 displayed on the console light will give, in its address section the address of the instruction responsible.



CHAPTER 7ADVANCED PROGRAMING TECHNIQUES

In the previous chapters we have covered the SWAP computer as far as needed in programing for SWIP. These functions are only a part of the total operations of SWAP. Through the preceding chapters we have seen how the computer may be employed to solve standard engineering problems. The method used in the solution was not the basic computer method. We have covered such concepts as mnemonic codes, symbolic addresses, parameters and numerical input. All these are not primarily attributes of the computer itself, but rather are features of the SWAP conversion program. After this conversion, the calculations by using SWIP are carried out on so-called floating point numbers. Thus, the two programs SWAP and SWIP compose a helpful tool for expediting the solution of engineering problems. At this time we can go into the study of the 650 computer itself.

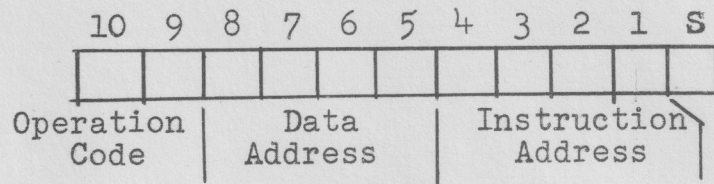
The 650 Computer - The 650 Computer is thoroughly described in the I.B.M. manual attached. We will, thus, describe only the very basic points of its operation. The 650 has, as mentioned in the introduction, all the basic computer elements: memory, arithmetic element, control and in-out elements. The arithmetic element is composed of a double length accumulator divided into two, ten digit length registers, the upper accumulator or the first ten most significant digits and the

lower accumulator, or the ten least significant digits. The arithmetic element also includes a ten digit signed register called the distributor. The basic difference between these registers and the simulated SWIP accumulator is the fact that in the 650 each of the three registers is addressable and thus may contain an instruction which can be performed. The 650 has also 2000 storage registers on a drum, each of which may contain a word; that is, in a combination of ten decimal digits and a sign. This word may be an instruction or a number.

Numbers - The 650 operates on a number of ten digits and a sign. In all arithmetic operations, these numbers are assumed to be fractions with the decimal point just to the left of the register. This type of a number is called fixed-point number. There, the number of significant digits will vary with the magnitude of the number. Thus, we can readily see one of the primary advantages of SWIP. In SWIP the numbers used preserved the same number of significant digits independent of the magnitude of the number, a most useful feature in engineering work. Such numbers are called floating point numbers. If we desired to make computations in basic 650 on data whose magnitude was greater or equal to one, we would have to scale them down by a chosen factor so that at no time during computations would the number exceed or be equal to one. In engineering work this is a serious handicap, completely eliminated by the use of SWIP.



Instructions - A 650 instruction is composed of a two digit operation code and two, four digit addresses in the form:



In most cases, the data address designates, as in SWIP instructions, where the data is to be located for the operation designated by the operation code. (See instruction chart in Appendix.) For some operations the data address signifies a certain number connected with the operation. The sign of the instruction is ignored. The instruction address signifies to the computer where the next instruction is to be taken from. Thus, unlike SWIP, the basic 650 is not sequential in performing instruction. The advantage of such a system lies in the fact that the time required to perform most 650 instructions is very small compared to the time required for the drum to complete one revolution. Thus, by properly selecting the instruction and data addresses in a program, the speed of solution may be substantially increased. The process of selecting proper addresses is called optimization of a program, and it is done automatically by SWAP.

The basic 650 has no index registers, but address modification, unlike SWIP, may be done by performing arithmetic operations on instructions. In order to be able, during a run,

to go from SWIP to basic 650 instruction, or vice versa, the following instruction is available in SWIP.

OUT A1 - Out of SWIP - Take the next instruction  
from location A1 and  
continue from there taking  
the instruction in A1 as a  
basic 650 instruction.

This instruction is a SWIP instruction and may have an index designation. Together with the IN instruction described previously, it enables the programmer to mix SWIP with basic 650.



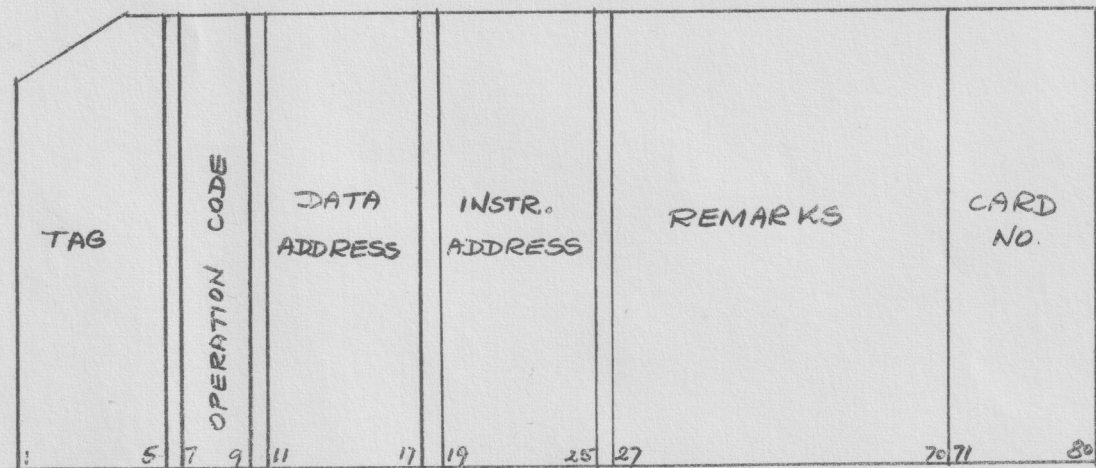
CHAPTER 8ADVANCED SWAP OPERATION AND FORMAT

From the previous chapter we have seen that SWAP converts regular machine code in addition to SWIP instruction. The basic difference between these two types of information lies in the fact that while SWIP is sequential, basic 650 code is not. In order to be able to convert both types, they have to be converted separately. Thus, SWAP requires two full passes over the program for conversion. The first of which is done while reading the alphabetic deck.

The alphabetic deck is read-in by SWAP in sections according to the availability of temporary storage inside the computer. Each card read is partially converted, made into a logical word, and then stored in the temporary storage. When the temporary storage fills up, it is punched out on cards and more alphabetic cards are then read in. This process continues until an END card is encountered, at which point SWAP completes the first pass. The arrangement of the cards at the start of conversion is such that first the SWAP first pass program is read in, followed by the alphabetic program deck, which in turn, is followed by the SWAP second pass program. The cards punched during the first pass should then follow the SWAP second pass program. Thus, when the first pass is completed, the second pass program is read in and, in turn, reads in the first pass output block by block. The number of cards in a block again

depends on the availability of temporary storage. When a block is read in, it is converted to its final numerical form and punched out. This process continues until the partially converted END card is read in, at which point the last block of cards is punched out followed by the symbol table and, if so desired, by a deck of availability cards. (See below.)

In Chapter 5 we covered certain aspects of the program format. Some more operation codes are available to permit conversion of a basic 650 coded program. In such a program the instruction format is as follows:

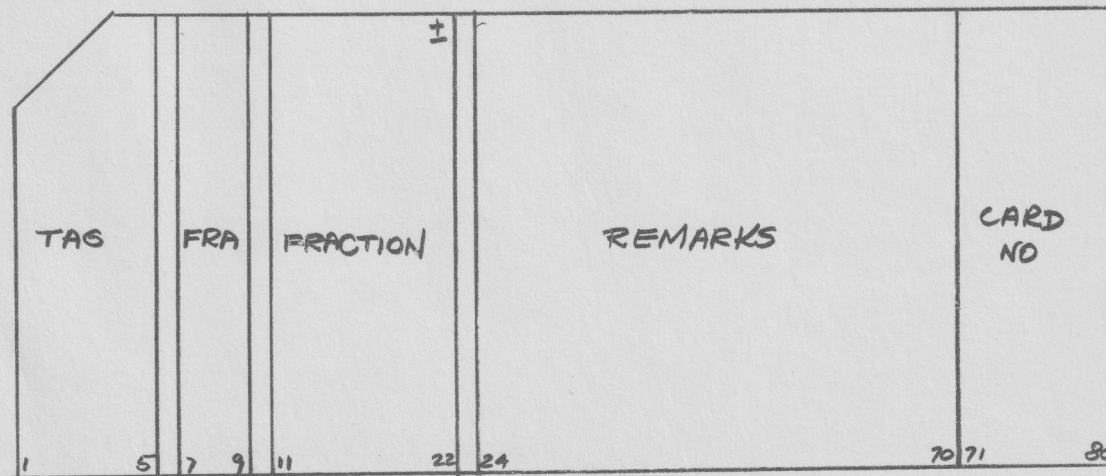
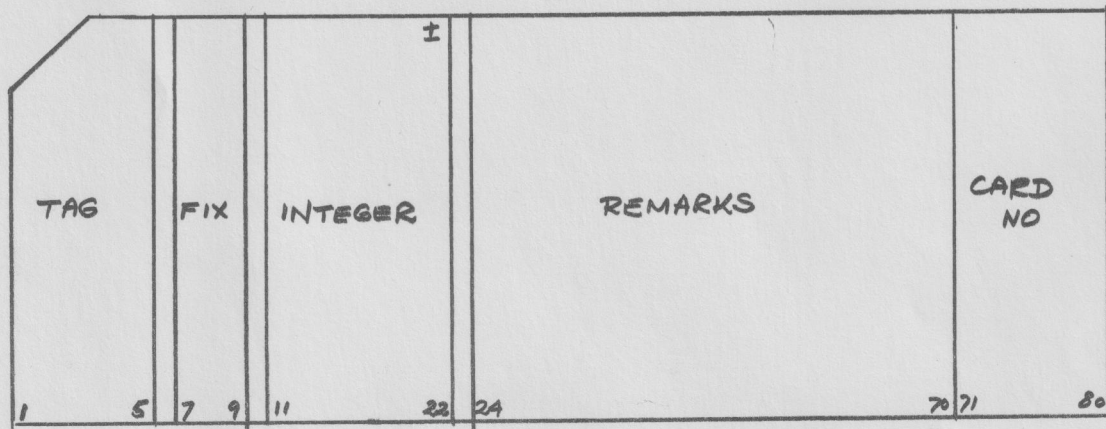


The tag has the same form as the SWIP instruction format and the data address and instruction address are of the same form as the SWIP address. However, since the program is not sequential, if no data address is specified, an optimum available location will be computed by SWAP. The instruction following will then be assigned to this location. Thus, if no instruction address is specified, the tag of the following



instruction should not be absolute or relative. A symbolic data address whose symbol was not previously defined, will be converted to the best optimum location available and the symbol will be given that equivalent. This will not be done with instructions whose data address does not refer to a storage location. In this case the symbol would have to be previously defined.

In order to read a fixed point number into the computer, two formats are made available.



Both formats consist of a set of numbers punched in the mantissa field. In the FRA card, the decimal point is assumed to be just to the left of the first digit. The number will be rounded to ten digits in that form. In the fix card, the decimal point is assumed to be just to the right of the last digit. This number will be multiplied by  $10^{-10}$  before being stored. Thus, the number FRA + 1234, will be stored as

|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S |
| 1  | 2 | 3 | 4 | 0 | 0 | 0 | 0 | 0 | 0 | + |

while the number, FIX + 1234, will be stored as

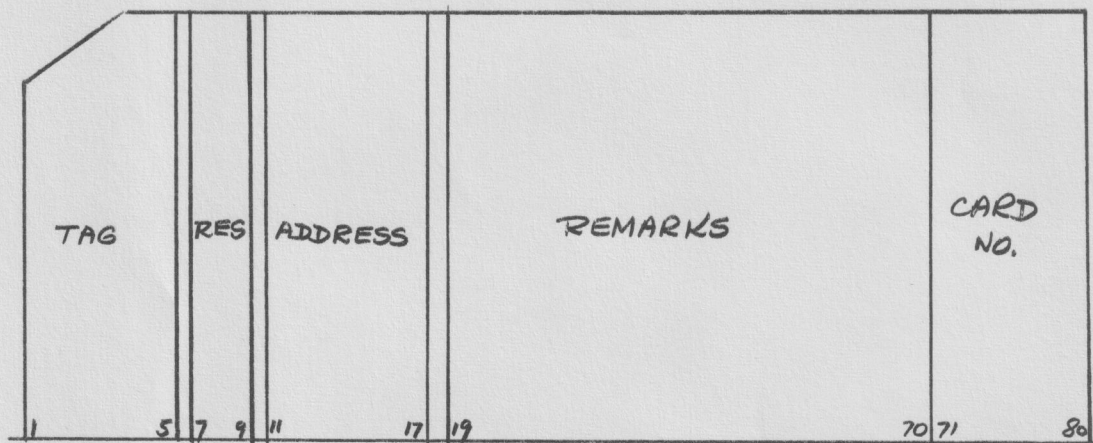
|    |   |   |   |   |   |   |   |   |   |   |
|----|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | S |
| 0  | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 3 | 4 | + |

Thus, we are able to load a register from either end. In both cases the sign has to be punched in column 22. Since the data address of instructions is given an unsequential optimum location, it is necessary to specify to SWAP when a number card with a symbolic tag should be placed in sequence. This can be done by a 12 punch in column 10. Parameters may be used in the mantissa field of either FRA or FIX cards.

Since all basic 650 information is placed in unsequential locations, it is necessary for SWAP to create an availability table. This table lists all locations in storage not yet filled with information. The availability table may be punched out at the end of the conversion. This permits conversion of a number of programs which are to be run together

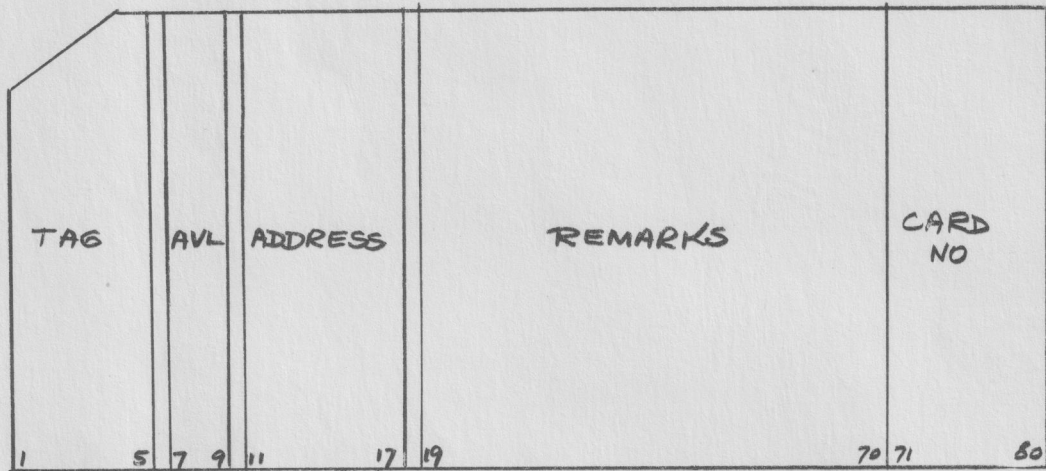


without overlapping. The designation requesting availability table punchout is made on the HED card. These availability cards may then be placed right after the HED card of the next program. It is often necessary to reserve certain chosen locations in storage for some future program. This can be done by a RES card.



The tag of this card should conform to the same rules as a SWIP instruction card, except that no parameter may be used. The data address, if symbolic, must be previously assigned. The RES card will reserve all locations starting from the one specified by the tag up to and including the location specified by the data address.

When it is desired to make a certain region on the drum available to the program, an AVL card is used.



This card operates in exactly the same manner as the RES card except that where the first reserves a block of locations, the AVL card makes it available.

In order to assign proper locations to words, SWAP during the first pass includes, besides the availability table, a program counter. This counter is set to zero at the start of the program and assumes any value up to a maximum depending on availability of storage due to the specification on the HED card. Each time a card of the type listed below is read in, the location for that word is assigned from the program counter and the program counter is increased by one. This location is also reserved in the availability table.

- a) SWIP instruction with no tag
- b) SWIP instruction with symbolic tag where symbol was not previously assigned



- c) FLO number
- d) Tagged FIX or FRA numbers with 12 punch in column 10
- e) Tagged FIX or FRA number followed by an untagged FIX or FRA number
- f) Untagged FIX or FRA number

If the above are tagged symbolically, the symbol will be assigned the value of the program counter.

If the cards listed below are read, the whole region specified is reserved and the program counter is increased by one more than the number of words effected.

- a) RES with previously unassigned symbolic tag
- b) DIT card

When an AVL card with a previously undefined symbol is read, the region will be made available, but the program counter will remain unchanged.

SWAP, in its first pass, also includes an LSR register. This register contains the last symbolic or absolute tag encountered and serves to evaluate relative addresses. All symbolic or absolute tags aside from an EQU or RES card, will be placed in the LSR register as they are read. All relative addresses will be computed relative to the contents of the LSR register aside from the DIT card address which will be computed relative to the location of the previous word.

When one of the following cards is read in, the program counter will be assigned a value from its tag. The

word will then be placed in that location, and the location will be reserved.

- a) A SWIP instruction with absolute relative or previously defined symbolic tag
- b) A FIX, FRA or FLO number with absolute or relative tag

The same will occur with an AVL card, with an absolute or relative tag, but no reservation will be made.

In case of DIT or RES cards with absolute or relative addresses, the program counter will be advanced to a value one more than the address and the appropriate reservation will be made. All symbolic tags read during the first pass are placed in the symbol table except in the following cases:

- a) Basic 650 instruction
- b) FIX or FRA number with no 12 punch in column 10 if not followed by an untagged FIX or FRA number

Aside from these cases, the symbol is assigned an equivalent from the program counter. In these two cases, no allocation is made and the same applies to any basic 650 instructions.

During the second pass, all symbols are converted. If a symbol is not found in the symbol table, a zero will be assumed and an indication to that effect made on the symbol table. During the second pass, all the words which were not assigned to a location in the first pass will be given an



optimum location from the availability table. Thus, priority is given by SWAP to all sequential information and the rest receives the locations which then remain available.

The alphabetic operation code is converted during the first pass to its numerical equivalent.

Several errors might occur during conversion. To eliminate these, several precautions should be observed. The program counter should not be allowed to exceed its limits set by the HED card. There must be sufficient storage space for all words. No illegal tag should be read (except as mentioned below). If relative addresses are used a symbolic or absolute tag must precede them. The symbol table must have sufficient space to accommodate all symbols, etc. (For a list of all possible alarms, see Appendix.)

Certain possible errors will not cause an alarm. Those are primarily due to an unassigned or duplicated symbol or to improper mixing of SWIP and basic 650 words so that an overlap occurs. Thus, if a basic 650 word is tagged with an absolute tag which denotes a location already assigned to a SWIP word, the SWIP word will be covered by the new word, thus resulting in error. To eliminate this condition, it is advisable to arrange the program so that all SWIP words are ahead of basic 650 words and to use as few absolute addresses as possible. This can be achieved by proper use of the RES and AVL cards, proper order during read-in and extensive use of symbols.

In order that optimization be done properly it is advisable to write the program so that the basic 650 instructions are ahead of the data. When an instruction with the instruction address of 8001, 8002 or 8003 is used, it is advisable to follow it by the instruction to be located at that address during the run of the numerical deck and tag it with that address. This will serve to optimize that instruction. The instructions tagged with 8001, 8002 or 8003 will not be converted, but their symbols will receive the proper optimum equivalent. No other illegal addresses may appear in the tag field aside from the EQU card.



APPENDIX

AVAILABLE STORAGE CHART

| <u>Routine</u>                    | <u>Allowable<br/>Program Counter</u> | <u>Number of<br/>Available Registers</u> |
|-----------------------------------|--------------------------------------|------------------------------------------|
| None                              | 0-1999                               | 2000                                     |
| SWIP                              | 0-1739                               | 1740                                     |
| IPC (and SWIP)                    | 0-1724                               | 1725                                     |
| IRD (and SWIP)                    | 0-1692                               | 1708                                     |
| IRD + IPC (and SWIP)              | 0-1692                               | 1693                                     |
| Read-in                           | 28-1999                              | 1972                                     |
| SWIP + Read-in                    | 28-1739                              | 1712                                     |
| IPC + Read-in (and SWIP)          | 28-1724                              | 1697                                     |
| IRD + Read-in (and SWIP)          | 28-1692                              | 1680                                     |
| IRD + IPC + Read-in<br>(and SWIP) | 28-1692                              | 1665                                     |



# III

## SWIP

Note: Symbols used are not legal SWAP symbols.  
IN = RAL (its own address) 1913

|    |     |      |      |                  |      |            |
|----|-----|------|------|------------------|------|------------|
| Y2 | LDD | UA1  |      | Enter from IN    | 1913 | 6919181924 |
|    | SDA | U1   | Y1   |                  | 1924 | 2218271974 |
| Y1 | RAL | U1   |      | Start new instr. | 1974 | 6518271815 |
|    | ALO | T8   | P2   |                  | 1815 | 1518181823 |
| P2 | STL | U1   | 8002 |                  | 1823 | 2018278002 |
| S6 | LDD | AB1  |      |                  | 1740 | 6917431746 |
|    | SDA | AA1  |      |                  | 1746 | 2217491752 |
|    | STL | T1   |      |                  | 1752 | 2017681771 |
|    | RAU | 8002 |      |                  | 1771 | 6080021779 |
|    | SRT | 0001 |      |                  | 1779 | 3000011785 |
|    | SUP | 8003 |      |                  | 1785 | 1180031793 |
|    | BMI | X1   |      | Index instr.?    | 1793 | 4617961747 |
|    | NZA | S1   | S2   | No index?        | 1747 | 4517501744 |
| S1 | SRT | 0005 |      |                  | 1750 | 3000051763 |
|    | SLO | T2   |      |                  | 1763 | 1617661821 |
|    | BMI | S7   | 4000 | Illegal index?   | 1821 | 4617744000 |
| S7 | ALO | T3   | 8002 |                  | 1774 | 1517778002 |
| S2 | ALO | AA1  |      |                  | 1744 | 1517491753 |
|    | SDA | A1   |      |                  | 1753 | 2217571760 |
| S3 | RAB | T1   |      |                  | 1760 | 6717681773 |
| S9 | SLT | 0002 |      |                  | 1773 | 3500021780 |
|    | SUP | T4   |      |                  | 1780 | 1117831837 |
|    | BMI | S5   | 3000 | Illegal instr.?  | 1837 | 4617903000 |
| S5 | AUP | T5   | 8003 | To jump table    | 1790 | 1017458003 |
| S4 | LDD | T6   | A2   | IRA              | 1861 | 6918161927 |
|    | LDD | M1   | A2   | IRS              | 1862 | 6917651927 |
|    | LDD | M2   | A2   | IRM              | 1863 | 6918171927 |
|    | LDD | M3   | A2   | IEX              | 1864 | 6917671927 |
|    | LDD | M4   | B2   | IST              | 1865 | 6918191972 |
|    | LDD | M5   | A2   | IAD              | 1866 | 6917691927 |
|    | LDD | M6   | A2   | ISU              | 1867 | 6918201927 |
|    | LDD | M7   | A2   | IAM              | 1868 | 6917721927 |
|    | LDD | M8   | A2   | ISM              | 1869 | 6918221927 |
|    | LDD | M9   | A2   | IDV              | 1870 | 6919231927 |
|    | LDD | M10  | A2   | IMR              | 1871 | 6918241927 |
|    | LDD | M11A | C17  | ITA              | 1872 | 6919781931 |
| S8 | LDD | U1   | C16  | IJU              | 1873 | 6918271891 |
|    | RAL | Z1   | C13  | IJN              | 1874 | 6519211834 |
|    | RSL | Z1   | C13  | IJP              | 1875 | 6619211834 |
|    | RAL | Z2   | C14  | IJZ              | 1876 | 6517981953 |
|    | SRT | 0004 | 8002 | OUT              | 1877 | 3000048002 |
|    | RAL | F1   | G1   | ICR              | 1878 | 6517411948 |
|    | RAL | F1   | G4   | ICT              | 1879 | 6517411946 |
|    | RAL | AA1  | H1   | ICI              | 1880 | 6517491941 |
|    | RAL | AA1  | AH1  | ICD              | 1881 | 6517491998 |

IV

|     |     |      |      |                   |      |            |
|-----|-----|------|------|-------------------|------|------------|
| A2  | STD | A3   | 8002 | Reset and add     | 1927 | 2418308002 |
| A4  | SLT | 0002 |      | to AR             | 1846 | 3500021803 |
|     | SLT | Z3   |      |                   | 1803 | 2018071810 |
|     | RAB | 8003 |      |                   | 1810 | 6780031917 |
|     | SLO | N2   |      |                   | 1917 | 1619201830 |
| A3  | STL | Z4   |      | Exit              | 1830 | 2018350000 |
| B2  | SDA | B1   |      | Store AC          | 1972 | 2219281781 |
|     | RAL | NA2  |      |                   | 1781 | 6517861794 |
|     | ALO | Z2   |      |                   | 1794 | 1517981853 |
|     | BMI | B3   | B4   |                   | 1853 | 4617561857 |
| B3  | STU | TA1  | B5   |                   | 1756 | 2118111814 |
| B4  | STL | TA1  | B5   |                   | 1857 | 2018111814 |
| B5  | RAL | Z1   |      |                   | 1814 | 6519211784 |
|     | BMI | B6   | B7   |                   | 1784 | 4617581808 |
| B6  | SUP | TA1  | B8   |                   | 1758 | 1118111915 |
| B7  | AUP | TA1  | B8   |                   | 1808 | 1018111915 |
| B8  | SRT | 0002 |      |                   | 1915 | 3000021922 |
|     | NZU | X1X  |      | Exponent too big? | 1922 | 4418421928 |
| B1  | STL |      |      | Exit              | 1928 | 2000000000 |
| C1  | STD | Z2   |      | IRA               | 1839 | 2417981802 |
|     | RAL | Z3   | P1   |                   | 1802 | 6518071812 |
| P1  | STL | Z1   | Y1   |                   | 1812 | 2019211974 |
| C2  | STD | Z2   |      | IRS               | 1838 | 2417981801 |
|     | RSL | Z3   | P1   |                   | 1801 | 6618071812 |
| C3  | STD | Z2   |      | IRM               | 1792 | 2417981754 |
|     | RAB | Z3   | P1   |                   | 1754 | 6718071812 |
| C4  | RAL | A1   |      | IEX               | 1804 | 6517571761 |
|     | LDD | T7   | B2   |                   | 1761 | 6917641972 |
|     | LDD | Z4   | C1   |                   | 1782 | 6918351839 |
| C6  | RAL | Z3   | D10  | IAD               | 1888 | 6518071762 |
| C7  | RSL | Z3   | D10  | ISU               | 1889 | 6618071762 |
| C8  | RAB | Z3   | D10  | IAM               | 1939 | 6718071762 |
| C9  | RSB | Z3   | D10  | ISM               | 1840 | 6818071762 |
| D10 | STL | ZA3  |      | Addition          | 1762 | 2019671832 |
|     | RSL | Z4   |      |                   | 1832 | 6618351845 |
|     | ALO | Z2   |      |                   | 1845 | 1517981903 |
|     | BMI | D1   | D2   |                   | 1903 | 4618061907 |
| D2  | SLO | N3   |      |                   | 1907 | 1618601965 |
|     | BMI | D11  | Y1   | Exp. difference   | 1965 | 4619681974 |
| D11 | LDD | Z1   |      | larger than 8     | 1968 | 6919211825 |
|     | STD | ZA1  |      |                   | 1825 | 2418821885 |
|     | SLT | 0004 |      |                   | 1885 | 3500041799 |
|     | ALO | NA3  |      |                   | 1799 | 1518051759 |
|     | STL | 'D6  |      |                   | 1759 | 2019751929 |
|     | RAL | ZA1  |      |                   | 1929 | 6518821937 |
|     | SRD | 0001 |      |                   | 1937 | 3100011843 |
|     | STL | ZB1  |      |                   | 1843 | 2018551914 |
|     | RAL | ZA3  |      |                   | 1914 | 6519671975 |



V

|     |     |      |     |              |      |            |
|-----|-----|------|-----|--------------|------|------------|
| D6  | SRD |      |     |              | 1975 | 3100001852 |
| D8  | ALO | ZB1  |     |              | 1852 | 1518551809 |
|     | RAU | 8002 |     |              | 1809 | 6080021919 |
|     | NZU | D8   | D7  | Answer is 0? | 1919 | 4418261925 |
| D8  | SCT |      |     |              | 1826 | 3600001847 |
|     | STU | Z1   |     |              | 1847 | 2119211926 |
|     | RSB | 8002 |     |              | 1926 | 6880021935 |
|     | ALO | T13  |     |              | 1935 | 1519381893 |
|     | ALO | Z2   |     |              | 1893 | 1517981770 |
| D12 | STL | Z2   | Y1  |              | 1770 | 2017981974 |
| D7  | RAL | NA1  | D12 |              | 1925 | 6517781770 |
| D1  | RAL | Z1   |     |              | 1806 | 6519211791 |
|     | LDD | ZA3  |     |              | 1791 | 6919671859 |
|     | STD | Z1   |     |              | 1859 | 2419211858 |
|     | STL | ZA3  |     |              | 1858 | 2019671831 |
|     | RSL | Z2   |     |              | 1831 | 6617981856 |
|     | ALO | Z4   |     |              | 1856 | 1518351841 |
|     | STD | Z2   | D2  |              | 1841 | 2417981907 |
| C10 | RAU | Z1   |     | LDV          | 1850 | 6019211976 |
|     | SRT | 0001 |     |              | 1976 | 3000011833 |
|     | NZA | D9   | D7  | Zero?        | 1833 | 4518961925 |
| D9  | DVR | Z3   |     |              | 1896 | 6418071848 |
|     | RAU | 8002 |     |              | 1848 | 6080021957 |
|     | SCT |      |     |              | 1957 | 3600001963 |
|     | STU | Z1   |     |              | 1963 | 2119211977 |
|     | RSB | 8002 |     |              | 1977 | 6880021985 |
|     | ALO | T13  |     |              | 1985 | 1519381943 |
|     | ALO | Z2   |     |              | 1943 | 1517981908 |
|     | SLO | Z4   | D12 |              | 1908 | 1618351770 |
| C11 | RAU | Z3   |     | IMR          | 1854 | 6018071964 |
|     | MPY | Z1   |     |              | 1964 | 1919211892 |
|     | RAU | 8003 |     |              | 1892 | 6080031849 |
|     | NZU | K2   | D7  | Zero?        | 1849 | 4419041925 |
| K2  | SCT |      |     |              | 1904 | 3600001911 |
|     | STU | Z1   |     |              | 1911 | 2119211932 |
|     | RSB | 8002 |     |              | 1932 | 6880021894 |
|     | ALO | Z2   |     |              | 1894 | 1517981930 |
|     | ALO | Z4   | D12 |              | 1930 | 1518351770 |
| C17 | SDA | M11  |     | ITA          | 1931 | 2218861989 |
|     | LDD | MA12 |     |              | 1989 | 6919421945 |
|     | SDA | M12  |     |              | 1945 | 2218991902 |
|     | RAB | T1   |     |              | 1902 | 6717681890 |
|     | SRT | 0001 |     |              | 1890 | 3000011999 |
|     | RAU | 8002 |     |              | 1999 | 6080021912 |
|     | SRT | 0003 |     |              | 1912 | 3000031742 |
|     | RAL | 8002 |     |              | 1742 | 6580021950 |

# VI

|     |     |      |      |
|-----|-----|------|------|
|     | SRT | 0003 |      |
|     | ALO | Z3   |      |
| M11 | LDD |      |      |
| M12 | SDA | 0000 | Y1   |
| C16 | STD | Z3   |      |
|     | SDA | U1   | 8001 |
| C13 | BMI | C12  | Y1   |
| C12 | RAL | A1   | S8   |
| C14 | ALO | NB2  | C13  |
| X1  | NZA | X2   | 6000 |
| X2  | SRT | 0005 |      |
|     | ALO | T2A  |      |
|     | BMI | X2X  | X4   |
| X4  | SLO | T3A  |      |
|     | SDA | F1   |      |
|     | RSL | T11  |      |
|     | AAB | T1   |      |
|     | BMI | X3X  | Q3   |
| Q3  | RAL | F1   |      |
|     | LDD | GA5  |      |
|     | SDA | G5   | S3   |
| G4  | LDD | GA8  |      |
|     | SDA | G8   |      |
|     | ALO | T12  |      |
|     | LDD | GA6  |      |
|     | SDA | G6   |      |
|     | RAB | T1   |      |
|     | SRT | 0001 |      |
|     | RAU | 8002 |      |
|     | SRT | 0003 |      |
|     | SUP | 8003 |      |
|     | SRT | 0003 | G5   |
| G5  | ALO |      |      |
| G11 | STL | TB1  |      |
| G6  | SLO |      |      |
| G13 | BMI | G7   | G9   |
| G7  | LDD | TB1  | G8   |
| G9  | LDD | NB1  | G8   |
| G8  | STD |      |      |
| G12 | BMI | G10  | Y1   |
| G10 | RAL | AA1  | S8   |
| G1  | LDD | GA2  |      |
|     | SDA | G2   |      |
|     | ALO | T12  |      |
|     | LDD | GA3  |      |
|     | SDA | G3   |      |
|     | LDD | NB1  |      |
| G2  | STD |      |      |

IJU

IJN IJP

IJZ

No index?

Illegal index?

Illegal instr.?

ICT

ICR

|      |            |
|------|------------|
| 1950 | 3000031961 |
| 1961 | 1518071886 |
| 1886 | 6900001899 |
| 1899 | 2200001974 |
| 1891 | 2418071960 |
| 1960 | 2218278001 |
| 1834 | 4618951974 |
| 1895 | 6517571873 |
| 1953 | 1519061834 |
| 1796 | 4518006000 |
| 1800 | 3000051813 |
| 1813 | 1519161971 |
| 1971 | 4619961776 |
| 1776 | 1618291887 |
| 1887 | 2217411844 |
| 1844 | 6617971751 |
| 1751 | 1717681775 |
| 1775 | 4619971836 |
| 1836 | 6517411795 |
| 1795 | 6917481851 |
| 1851 | 2217551760 |
| 1946 | 6919491952 |
| 1952 | 2219051958 |
| 1958 | 1519621969 |
| 1969 | 6919791883 |
| 1883 | 2219541910 |
| 1910 | 6717681980 |
| 1980 | 3000011988 |
| 1988 | 6080021898 |
| 1898 | 3000031909 |
| 1909 | 1180031966 |
| 1966 | 3000031755 |
| 1755 | 1500001897 |
| 1897 | 2019011954 |
| 1954 | 1600001987 |
| 1987 | 4619401947 |
| 1940 | 6919011905 |
| 1947 | 6919001905 |
| 1905 | 2400001992 |
| 1992 | 4619951974 |
| 1995 | 6517491873 |
| 1948 | 6919511955 |
| 1955 | 2219731990 |
| 1990 | 1519621970 |
| 1970 | 6919341994 |
| 1994 | 2219591936 |
| 1936 | 6919001973 |
| 1973 | 2400001944 |



# VII

|      |     |      |       |
|------|-----|------|-------|
| G14  | RAL | AA1  |       |
|      | LDD | N1   |       |
| G3   | SDA | 0000 | Y1    |
| H1   | SDA | TB1  |       |
|      | RAL | F1   |       |
|      | LDD | GA3  |       |
|      | SDA | H2   |       |
| Q2   | RAL | TB1  |       |
| F1   | ALO |      |       |
| H2   | SDA | 0000 | Y1    |
| AH1  | LDD | T10  | H1    |
| UA1  | RAL | 0000 | S6    |
| U1   | RAL | 0000 | S6    |
| T8   | NOP | 0001 |       |
| AB1  | RAL | 0000 | A4    |
| AA1  | RAL | 0000 | A4    |
| A1   | RAL | 0000 | A4    |
| T1   | NOP |      |       |
| T2   | NOP | 0004 |       |
| T3   | RAL | 11+3 | S2    |
| I1   | NOP |      |       |
|      | NOP |      |       |
|      | NOP |      |       |
| T4   | NOP | 0000 | 0021  |
| T5   | RAL | A1   | S4+21 |
| T6   | STL | Z4   | C1    |
| M1   | STL | Z4   | C2    |
| M2   | STL | Z4   | C3    |
| M3   | STL | Z4   | C4    |
| M4   | STL | 0000 | Y1    |
| M5   | STL | Z4   | C6    |
| M6   | STL | Z4   | C7    |
| M7   | STL | Z4   | C8    |
| M8   | STL | Z4   | C9    |
| M9   | STL | Z4   | C10   |
| M10  | STL | Z4   | C11   |
| M11A | LDD | 0000 | M12   |
| Z3   | NOP |      |       |
| N2   | NOP | 0000 | 0050  |
| Z4   | NOP |      |       |
| NA2  | NOP | 0000 | 0050  |
| Z2   | NOP |      |       |
| TA1  | NOP |      |       |
| Z1   | NOP |      |       |
| X1X  | NOP | 0000 | 5000  |
| T7   | STL | 0000 | 1782  |
| ZA3  | NOP |      |       |
| N3   | NOP | 0000 | 0009  |

ICI

ICD

Program counter

Index 1  
Index 2  
Index 3

AR mantissa

AR exponent

MRA exponent

MRA mantissa

|      |            |
|------|------------|
| 1944 | 6517491956 |
| 1956 | 6919861959 |
| 1959 | 2200001974 |
| 1941 | 2219011984 |
| 1984 | 6517411828 |
| 1828 | 6919341991 |
| 1991 | 2219331993 |
| 1993 | 6519011741 |
| 1741 | 1500001933 |
| 1933 | 2200001974 |
| 1998 | 6918841941 |
| 1918 | 6500001740 |
| 1827 | 6500001740 |
| 1818 | 0000010000 |
| 1743 | 6500001846 |
| 1749 | 6500001846 |
| 1757 | 6500001846 |
| 1768 | 0000000000 |
| 1766 | 0000040000 |
| 1777 | 6517901744 |
| 1787 | 0000000000 |
| 1788 | 0000000000 |
| 1789 | 0000000000 |
| 1783 | 0000000021 |
| 1745 | 6517571882 |
| 1816 | 2018351839 |
| 1765 | 2018351838 |
| 1817 | 2018351792 |
| 1767 | 2018351804 |
| 1819 | 2000001974 |
| 1769 | 2018351888 |
| 1820 | 2018351889 |
| 1772 | 2018351939 |
| 1822 | 2018351840 |
| 1923 | 2018351850 |
| 1824 | 2018351854 |
| 1978 | 6900001899 |
| 1807 | 0000000000 |
| 1920 | 0000000050 |
| 1835 | 0000000000 |
| 1786 | 0000000050 |
| 1798 | 0000000000 |
| 1811 | 0000000000 |
| 1921 | 0000000000 |
| 1842 | 0000005000 |
| 1764 | 2000001782 |
| 1967 | 0000000000 |
| 1860 | 0000000009 |

## VIII

|      |     |       |      |
|------|-----|-------|------|
| ZA1  | NOP |       |      |
| NA3  | SRD | 0010  | D8   |
| ZB1  | NOP |       |      |
| T13  | NOP | 0000  | 0001 |
| NA1  | -00 | 0000  | 0050 |
| MA12 | SDA | 0000  | Y1   |
| NB2  | NOP | 0000  | 0049 |
| T2A  | NOP | 0003  |      |
| X2X  | NOP | 0000  | 4000 |
| T3A  | ALO | I1+2  | H2   |
| T11  | AAB |       |      |
| X3X  | NOP | 0000  | 3000 |
| GA5  | ALO | 0000  | G11  |
| GA8  | STD | 0000  | G12  |
| T12  | NOP | I2-I1 |      |
| I2   | NOP |       |      |
|      | NOP |       |      |
|      | NOP |       |      |
| GA6  | SLO | 0000  | G13  |
| TB1  | NOP |       |      |
| NB1  | NOP |       |      |
| GA2  | STD | 0000  | G14  |
| GA3  | SDA | 0000  | Y1   |
| N1   | NOP |       |      |
| T10  | -00 | 0000  | 0000 |

Criterion 1  
 Criterion 2  
 Criterion 3

|      |             |
|------|-------------|
| 1882 | 0000000000  |
| 1805 | 3100101852  |
| 1855 | 0000000000  |
| 1938 | 0000000001  |
| 1778 | -0000000050 |
| 1942 | 2200001974  |
| 1906 | 0000000049  |
| 1916 | 0000030000  |
| 1996 | 0000004000  |
| 1829 | 1517891933  |
| 1797 | 1700000000  |
| 1997 | 0000003000  |
| 1748 | 1500001897  |
| 1949 | 2400001992  |
| 1962 | 0001940000  |
| 1981 | 0000000000  |
| 1982 | 0000000000  |
| 1983 | 0000000000  |
| 1979 | 1600001987  |
| 1901 | 0000000000  |
| 1900 | 0000000000  |
| 1951 | 2400001944  |
| 1934 | 2200001974  |
| 1986 | 0000000000  |
| 1884 | -0000000000 |



## IX

SWIPSequential

1740 6917431746  
1741 1500001933  
1742 6580021950  
1743 6500001846  
1744 1517491753  
1745 6517571882  
1746 2217491752  
1747 4517501744  
1748 1500001897  
1749 6500001846  
1750 3000051763  
1751 1717681775  
1752 2017681771  
1753 2217571760  
1754 6718071812  
1755 1500001897  
1756 2118111814  
1757 6500001846  
1758 1118111915  
1759 2019751929  
1760 6717681773  
1761 6917641972  
1762 2019671832  
1763 1617661821  
1764 2000001782  
1765 2018351838  
1766 0000040000  
1767 2018351804  
1768 0000000000  
1769 2018351888  
1770 2017981974  
1771 6080021779  
1772 2018351939  
1773 3500021780  
1774 1517778002  
1775 4619971836  
1776 1618291887  
1777 6517901744  
1778 -0000000050  
1779 3000011785  
1780 1117831837  
1781 6517861794  
1782 6918351839  
1783 0000000021  
1784 4617581808  
1785 1180031793

1786 0000000050  
1787 0000000000  
1788 0000000000  
1789 0000000000  
1790 1017458003  
1791 6919671859  
1792 2417981754  
1793 4617961747  
1794 1517981853  
1795 6917481851  
1796 4518006000  
1797 1700000000  
1798 0000000000  
1799 1518051759  
1800 3000051813  
1801 6618071812  
1802 6518071812  
1803 2018071810  
1804 6517571761  
1805 3100101852  
1806 6519211791  
1807 0000000000  
1808 1018111915  
1809 6080021919  
1810 6780031917  
1811 0000000000  
1812 2019211974  
1813 1519161971  
1814 6519211784  
1815 1518181823  
1816 2018351839  
1817 2018351792  
1818 0000010000  
1819 2000001974  
1820 2018351889  
1821 4617744000  
1822 2018351840  
1823 2018278002  
1824 2018351854  
1825 2418821885  
1826 3600001847  
1827 6500001740  
1828 6919341991  
1829 1517891933  
1830 2018350000  
1831 6617981856

1832 6618351845  
 1833 4518961925  
 1834 4618951974  
 1835 0000000000  
 1836 6517411795  
 1837 4617903000  
 1838 2417981801  
 1839 2417981802  
 1840 6818071762  
 1841 2417981907  
 1842 0000005000  
 1843 2018551914  
 1844 6617971751  
 1845 1517981903  
 1846 3500021803  
 1847 2119211926  
 1848 6080021957  
 1849 4419041925  
 1850 6019211976  
 1851 2217551760  
 1852 1518551809  
 1853 4617561857  
 1854 6018071964  
 1855 0000000000  
 1856 1518351841  
 1857 2018111814  
 1858 2019671831  
 1859 2419211858  
 1860 0000000009  
 1861 6918161927  
 1862 6917651927  
 1863 6918171927  
 1864 6917671927  
 1865 6918191972  
 1866 6917691927  
 1867 6918201927  
 1868 6917721927  
 1869 6918221927  
 1870 6919231927  
 1871 6918241927  
 1872 6919781931  
 1873 6918271891  
 1874 6519211834  
 1875 6619211834  
 1876 6517981953  
 1877 3000048002  
 1878 6517411948  
 1879 6517411946  
 1880 6517491941  
 1881 6517491998  
 1882 0000000000  
 1883 2219541910

1884 -0000000000  
 1885 3500041799  
 1886 6900001899  
 1887 2217411844  
 1888 6518071762  
 1889 6618071762  
 1890 3000011999  
 1891 2418071960  
 1892 6080031849  
 1893 1517981770  
 1894 1517981930  
 1895 6517571873  
 1896 6418071848  
 1897 2019011954  
 1898 3000031909  
 1899 2200001974  
 1900 0000000000  
 1901 0000000000  
 1902 6717681890  
 1903 4618061907  
 1904 3600001911  
 1905 2400001992  
 1906 0000000049  
 1907 1618601965  
 1908 1618351770  
 1909 1180031966  
 1910 6717681980  
 1911 2119211932  
 1912 3000031742  
 1913 6919181924  
 1914 6519671975  
 1915 3000021922  
 1916 0000030000  
 1917 1619201830  
 1918 6500001740  
 1919 4418261925  
 1920 0000000050  
 1921 0000000000  
 1922 4418421928  
 1923 2018351850  
 1924 2218271974  
 1925 6517781770  
 1926 6880021935  
 1927 2418308002  
 1928 2000000000  
 1929 6518821937  
 1930 1518351770  
 1931 2218861989  
 1932 6880021894  
 1933 2200001975  
 1934 2200001974  
 1935 1519381893



# XI

1936 6919001973  
 1937 3100011843  
 1938 0000000001  
 1939 6718071762  
 1940 6919011905  
 1941 2219011984  
 1942 2200001974  
 1943 1517981908  
 1944 6517491956  
 1945 2218991902  
 1946 6919491952  
 1947 6919001905  
 1948 6919511955  
 1949 2400001992  
 1950 3000031961  
 1951 2400001944  
 1952 2219051958  
 1953 1519061834  
 1954 1600001987  
 1955 2219731990  
 1956 6919861959  
 1957 3600001963  
 1958 1519621969  
 1959 2200001974  
 1960 2218278001  
 1961 1518071886  
 1962 0001940000  
 1963 2119211977  
 1964 1919211892  
 1965 4619681974  
 1966 3000031755  
 1967 0000000000  
 1968 6919211825  
 1969 6919791883  
 1970 6919341994  
 1971 4619961776  
 1972 2219281781  
 1973 2400001944  
 1974 6518271815  
 1975 3100101852  
 1976 3000011833  
 1977 6880021985  
 1978 6900001899  
 1979 1600001987  
 1980 3000011988  
 1981 0000000000  
 1982 0000000000  
 1983 0000000000  
 1984 6517411828  
 1985 1519381943  
 1986 0000000000

1987 4619401947  
 1988 6080021898  
 1989 6919421945  
 1990 1519621970  
 1991 2219331993  
 1992 4619951974  
 1993 6519011741  
 1994 2219591936  
 1995 6517491873  
 1996 0000004000  
 1997 0000003000  
 1998 6918841941  
 1999 6080021912

# XII

## IPC ROUTINE

IPC = 150 1730

|      |     |      |      |
|------|-----|------|------|
| 1725 | LDD | 1798 | 1726 |
| 1726 | STD | 1729 | 1737 |
| 1727 | NOP | 0000 | 0000 |
| 1728 | RSL | 7376 | 0000 |
| 1729 | NOP | 0000 | 0000 |
| 1730 | ITA | 1739 | 0010 |
| 1731 | IST | 1727 |      |
| 1732 | IRA | 1727 |      |
| 1733 | OUT | 1734 |      |
| 1734 | LDD | 1921 | 1735 |
| 1735 | STD | 1727 | 1725 |
| 1736 | NOP | 0000 | 0080 |
| 1737 | PCH | 1727 | 1738 |
| 1738 | IN  |      |      |
| 1739 | IJU |      |      |

Mantissa

FLO

Exponent

Entry

Exponent too big?

Control word

Exit

|            |
|------------|
| 6917981726 |
| 2417291737 |
| 0000000000 |
| 6673760000 |
| 0000000000 |
| 1117390010 |
| 0417270000 |
| 0017270000 |
| 1617340000 |
| 6919211735 |
| 2417271725 |
| 0000000080 |
| 7117271738 |
| 6517381913 |
| 1200000000 |



# XIII

## IRD ROUTINE

IRD = IJU 1720

|      |     |      |      |           |            |
|------|-----|------|------|-----------|------------|
| 1693 | RAL | 8002 | 1713 |           | 6580021713 |
| 1694 | RAL | 8002 | 1715 |           | 6580021715 |
| 1695 | RSL | 1701 | 1700 |           | 6617011700 |
| 1696 | STU | 1701 | 1717 |           | 2117011717 |
| 1697 | STU | 1701 | 1714 |           | 2117011714 |
| 1698 | RAL | 1701 | 1700 |           | 6517011700 |
| 1699 | RAU | 1702 | 1711 |           | 6017021711 |
| 1700 | STL | 1798 | 1723 |           | 2017981723 |
| 1701 | NOP | 0000 | 0000 |           | 0000000000 |
| 1702 | NOP | 0000 | 0000 |           | 0000000000 |
| 1703 | NOP | 0000 | 0000 |           | 0000000000 |
| 1704 | NOP | 0000 | 0000 |           | 0000000000 |
| 1705 | NOP | 0000 | 0000 | Read band | 0000000000 |
| 1706 | NOP | 0000 | 0000 |           | 0000000000 |
| 1707 | NOP | 0000 | 0000 |           | 0000000000 |
| 1708 | NOP | 0000 | 0000 |           | 0000000000 |
| 1709 | NOP | 0000 | 0000 |           | 0000000000 |
| 1710 | NOP | 0000 | 0000 |           | 0000000000 |
| 1711 | SRT | 0003 | 1693 |           | 3000031693 |
| 1712 | STD | 1921 | 1699 |           | 2419211699 |
| 1713 | SLT | 0001 | 1697 |           | 3500011697 |
| 1714 | SLT | 0001 | 1694 |           | 3500011694 |
| 1715 | AUP | 1701 | 1716 |           | 1017011716 |
| 1716 | SLT | 0001 | 1696 |           | 3500011696 |
| 1717 | RAL | 1708 | 1718 |           | 6517081718 |
| 1718 | BMI | 1695 | 1698 |           | 4616951698 |
| 1719 | LDD | 1709 | 1712 |           | 6917091712 |
| 1720 | ITA | 1724 | 0010 | Entry     | 1117240010 |
| 1721 | OUT | 1722 |      |           | 1617220000 |
| 1722 | RDS | 1722 | 1719 | Read card | 7017221719 |
| 1723 | IN  |      |      |           | 6517231913 |
| 1724 | IJU |      |      | Exit      | 1200000000 |

## XIV

READ IN ROUTINE

| LOC  | OP  | DA   | IA   |               |            |
|------|-----|------|------|---------------|------------|
| 0000 | NOP | 0001 | 0000 | Stop register | 0000010000 |
| 0001 | NOP | 0000 | 0000 |               | 0000000000 |
| 0002 | NOP | 0000 | 0000 |               | 0000000000 |
| 0003 | NOP | 0000 | 0000 |               | 0000000000 |
| 0004 | NOP | 0000 | 0000 |               | 0000000000 |
| 0005 | NOP | 0000 | 0000 | Read in band  | 0000000000 |
| 0006 | NOP | 0000 | 0000 |               | 0000000000 |
| 0007 | NOP | 0000 | 0000 |               | 0000000000 |
| 0008 | NOP | 0000 | 0000 |               | 0000000000 |
| 0009 | NOP | 0000 | 0000 |               | 0000000000 |
| 0010 | NOP | 0000 | 0000 |               | 0000000000 |
| 0011 | AUP | 0000 | 0018 |               | 1000000018 |
| 0012 | NOP | 0000 | 0000 |               | 0000000000 |
| 0013 | LDD | 0015 | 0019 |               | 6900150019 |
| 0014 | ALO | 8001 | 0023 |               | 1580010023 |
| 0015 | STD | 0000 | 0011 |               | 2400000011 |
| 0016 | NZU | 0020 | 0021 |               | 4400200021 |
| 0017 | RAL | 0024 | 0025 |               | 6500240025 |
| 0018 | ALO | 8001 | 0026 |               | 1580010026 |
| 0019 | SDA | 0015 | 0022 |               | 2200150022 |
| 0020 | AUP | 8001 | 8002 |               | 1080018002 |
| 0021 | RDS | 0027 | 0021 |               | 7000270021 |
| 0022 | SLT | 0004 | 0014 |               | 3500040014 |
| 0023 | SDA | 0012 | 0017 |               | 2200120017 |
| 0024 | LDD | 0002 | 8003 |               | 6900028003 |
| 0025 | AUP | 0015 | 8002 |               | 1000158002 |
| 0026 | SUP | 0012 | 0016 |               | 1100120016 |
| 0027 | RAL | 0001 | 0013 |               | 6500010013 |