APOLLO GUIDANCE COMPUTER

Information Series

ISSUE 6B

INTERPRETER AND INTERPRETIVE INSTRUCTIONS

FR-2-106B

29 May 1964

CONTENTS

CONTENTS (cont)

ILLUSTRATIONS

TABLES

ATTACHMENTS

6-1. INTRODUCTION

6-2. This is the sixth issue of the AGCIS, which is published to inform the technical staff at MIT/IL and Raytheon about the Apollo guidance computer (AGC) subsystem. In Issue 15 various AGC programs and program sections are described in paragraph 15-98 through the end. The Interpreter and Interpretive Instruction are discussed in paragraphs 15-6, 15-83, 15-84, 15-90 through 15-97, and 15-105. This issue contains a description of program section Interpreter as used in programs SUNRSE33 and SUNRSE38. The Interpreter was originally designed for the AGC3 by Dr. J. H. Laning of MIT/IL and later modified by C. A. Muntz also of MIT/IL for the AGC4.

6-3. An AGC program is defined as the entire content of F memory (paragraph 15-5) and consists of various program sections. Program sections may consists of several major routines, which are composed of minor routines and subroutines.

6-4. The Interpreter is a highly sophisticated program section designed to execute program portions written in interpretive language (paragraph 15-6). The Interpreter consists of approximately 1700 Regular Instructions and constants. The Interpreter is written in basic machine language and is able to translate and execute program portions written in interpretive language. Eleven Regular Machine Instructions (table 15-12) can be used to write basic program portions, and 71 Interpretive Instructions (table 15-13) with a total of 126 different order codes are provided for writing interpretive program portions.

6-5. Regular Instructions are single address instructions consisting of a 3-bit order code and a 12-bit relevant address both contained in the same word (figure 15-12). Interpretive Instructions are single or no address instructions consisting of a 7-bit order code and a 15-bit relevant address if necessary. A single address Interpretive Instruction is contained in 1-1/2 words; the order code in an Interpretive Instruction Word (IIW), the relevant address in an Interpretive Address Word (IAW) as shown in figure 15-12. A no address Interpretive Instruction is contained in half a word only, i. e. in an IIW.

6-6. Program portions expressed in basic language are written according to the rules of parenthesis-free notation. The instructions of

a basic subroutine are arranged in the sequence in which they have to be executed. Program portions expressed in interpretive language are also written according to the rules of parenthesis-free notation. (All program sections written either partly or completely in interpretive language are situated in banks 21 through 34 which are referred to as higher banks.) Interpretive subroutines are arranged in interpretive strings. Each interpretive string consists of a group of IIW's, (at least one IIW) referred to as an instruction string. Each instruction string is followed by a group of IAW's, (at least one IAW) referred to as an address string. The order codes of an interpretive string are arranged in the sequence in which the Interpretive Instructions have to be executed. (The order code in bit positions 14 through 8 precedes the order code in bit positions 7 through 1.) The relevant addresses are arranged within the address string in the same sequence as stated above without leaving spaces for non-address Interpretive Instructions.

6-7. INTERPRETIVE STRINGS

6-8. SIMPLE STRINGS

6-9. A basic routine for computing a+b-c = x with single precision may be written as follows:

```
CS   C
AD   B
AD   A
TS   X
```

Instruction CS C (table 15-12) clears the accumulator (address 00000) and enters the complement content of location C into it. Instruction AD B adds the content of location B to the content of the accumulator. Instruction AD A adds the content of location C to the latest content of the accumulator. Instruction TS X transfers the result contained in the accumulator to location X. The designations A, B, C, and X have no relation to the name of any location; they are only used to symbolize the addresses of those locations where quantities a, b, c, and x are stored.

6-10. An interpretive string for computing a+b-c = x with double precision may be written as follows:

```
DAD      1
DSU
         A
         B
         C
STORE    X
```

The instruction string of this interpretive string consists of two IIW's. Bits 7 through 1 of the first IIW of a string always represents an integer which indicates the number of IIW's immediately following the first IIW of a string (figure 15-12). Bits 14 through 8 of the first IIW represent the order code of the first Interpretive Instruction and bits 14 through 8 of the second IIW represent the order code of the second Interpretive Instruction. In the example discussed, bits 7 through 1 of the second IIW contain no order code. The address string consists

of four IAW's; the first represents a load address, the second and third operand addresses, and the fourth a store address.

6-11. Normally, the Interpreter loads the multiprecision accumulator (MPAC) in use or the vector accumulator (VAC) in use when it recognizes the beginning of an interpretive string. Since the Job Control (Executive, Issue 12) moves a Job for its execution into Job Area 1, only the MPAC of Job Area 1, (locations 0114 through 0116, table 15-15) is accessible during the actual execution. If a Job requires the use of a VAC, the Job Control assigns one out of five VAC's to the Job therefore, anyone of the five vector accumulators may be accessable during the execution of an Interpretive Program portion. In the example given, MPAC is loaded by instruction DAD K. Since order code DAD represents a Dual-Quantity Instruction (paragraph 15-94), this instruction enters the quantity located at the load address (A) and operates with the quantity located at the first operand address (B). After entering the content of location A (and A +1) into MPAC (and MPAC +1), the Interpreter adds the content of locations B (and B +1) to the content of MPAC (and MPAC +1). The expressions (and A + 1), (and MPAC +1), etc., indicate that the double-precision quantities are stored in two locations. Reference is normally made only to the first location of a double-precision quantity, of a triple-precision, or of a vector (paragraph 15-94).

6-12. Once the addition is complete, the Interpreter subtracts the content of location C (and C + 1) from the content of MPAC (and MPAC +1) and holds the final result in MPAC (and MPAC + 1). As the Interpreter recognizes word STORE X, it transfers the content of MPAC (and MPAC +1) to locations X (and X +1).

6-13. An interpretive string for computing

$$\frac{a + \cos\left(b + \sqrt{a^2 + b}\right)}{c} = x$$

with double precision can be writen as follows:

L	DSQ	3
L + 1	DAD	SQRT
L + 2	DAD	COS
L + 3	DAD	DDV
L + 4		A
L + 5		B
L + 6		B
L + 7		A
L + 8		C
L + 9	STORE	X

This string consists of one instruction string with four IIW's and one address string with six IAW's. When the Interpreter reads the word DSQ 3, it recognizes that three more instruction words follow to complete the IIW string. The Interpreter also recognizes that it has to take the first data from address A. Address A is stored at location L + 4, where L is the location of word DSQ 3. As the Interpreter decodes the first IIW, it clears the MPAC and enters the double-precision quantity a from location A into it. Since order code DSQ (table 15-13) represents a Single-Quantity Instruction (paragraph 15-95), the quantity a in MPAC is squared and no other quantity is needed for this operation. After squaring, the Interpreter recognizes that Interpretive Instruction DAD B has to be executed next. Since a^2 is now contained in the MPAC, the computer adds the quantity b to a^2 and the MPAC retains $a^2 + b$. At the next step the Interpreter recognizes that it has to obtain the square root of the MPAC content by executing the Single-Quantity Instruction SQRT. The MPAC then retains the quantity $\sqrt{a^2 + b}$. Next the Interpreter recognizes that it has to add the quantity b to the MPAC content; the MPAC retains $b + \sqrt{a^2 + b}$ after the addition. The fifth Interpretive Instruction to be executed, COS, is again of the single-quantity type. The MPAC then contains the quantity $\cos (b + \sqrt{a^2 + b})$. Quantity a is then added to the MPAC content by executing DAD A, as taken from locations L + 3 and L + 7. When DDV C is executed, the MPAC content is divided by quantity c and retained in the MPAC. Finally, the MPAC content is transferred to location X (and X+1).

6-14. STORING AND RECALLING PARTIAL RESULTS

6-15. When ab + cd = x is computed, it is necessary to store partial results before the final result can be calculated. If

```
DMP     0
        A
        B
STORE   Y
```

is executed, the double-precision product ab = y is stored at location Y. When the last address of the above string is omitted, i.e., when string

```
DMP     0
        A
        B
```

is executed, the Interpreter automatically puts the partial result (ab) into the next two free locations of the Pushlist in use. (Pushlists are a series of erasable registers reserved for temporary storage; refer

to Work Areas of table 15-15.) One of the two following methods can be applied to recall a quantity from a Pushlist: either supplying no operand address or supplying an inactive address.

6-16. If an instruction finds no address word in the address string, it automatically takes the operand from the proper location of the Pushlist. For example, consider the following routine, which consists of two interpretive strings (each consisting of an instruction string and an address string):

```
DMP     0
        A
        B
DMP     1
DAD     0
        C
        D
```

After quantity a is multiplied by quantity b, the product ab is stored in the Pushlist as described in paragraph 6-15. After quantity c is multiplied by quantity d, the Interpreter recognizes that it has to execute an addition. However, the Interpreter cannot find the address of the quantity to be added to the MPAC content (product cd). Therefore, it automatically takes the product ab, which was temporarily stored in the Pushlist. The final result, ab + cd = x, is again stored in the Pushlist because no address is given. In fact, the final result is stored in the same two locations where partial result ab was stored.

6-17. The Interpreter must distinguish between an address where the result is to be stored (store address) and an address where data is to be taken (operand address). This allows data to be recalled from the Pushlist before the result is stored elsewhere. Therefore, a special format is employed which is recognized by the Interpreter for a store address. The routine:

```
DMP     0
        A
        B
DMP     1
DAD     0
        C
        D
STORE   X
```

is very similar to the routine listed in paragraph 6-16, except that address word STORE X is added. STORE X specifies that address X is to be used for storing the final result. No address is given for

calling the operand when DAD is executed. For this reason the product ab is called from the Pushlist and added to the product cd. The final result is stored at location X.

6-18. A special code, Inactive Address (77777), references the Pushlist. A routine to compute:

$$a + \sqrt{a^2 + b^2} = x$$

can be written as follows:

```
DSQ     0
        B
DSQ     2
DAD     SQRT
DAD
        A

        A
STORE   X
```

After quantity b is squared and b^2 is placed into the Pushlist, quantity a is squared and a^2 is retained in the MPAC. Next, the Interpreter recognizes that DAD should be executed. Address word 77777 (symbolized by θ) tells the computer to take the quantity b^2 from the Pushlist locations last used and add it to the MPAC content a^2. After the square root of $a^2 + b^2$ is obtained, the quantity a is added to the quantity $\sqrt{a^2 + b^2}$ in the MPAC, and the final result is transferred to location X.

6-19. Sometimes more than one set of locations in the Pushlist is needed to store partial results. A routine to compute

$$\frac{a^2 + b^2}{c^2 + d^2} = x$$

can be written as follows:

```
DSQ     0
        D
DSQ     1
DAD
        C
DSQ     0
        B
DSQ     1
DAD     DDV
        A
STORE   X
```

This routine consists of four interpretive strings. First, d^2 is computed and stored in the Pushlist at location P (and P +1). Next, c^2 is calculated and remains in the MPAC. During the execution of the first DAD instruction, the Interpreter cannot find an operand address and adds the quantity d^2 in location P to quantity c^2. The sum $c^2 + d^2$ is again stored in location P. Thereafter, b^2 is computed and stored in location P +2 (and P +3). Next a^2 is computed and retained in the MPAC. During the execution of the second DAD instruction, the Interpreter cannot find an operand address and adds the quantity b^2 in location P +2 to the quantity a^2. Quantity $a^2 + b^2$ remains in the MPAC. The Interpreter then recognizes that instruction DDV has to be executed but cannot find an address word other than STORE X. For this reason the Interpreter takes quantity $c^2 + d^2$ from location P, divides $a^2 + b^2$ by $c^2 + d^2$, and stores the quotient at X.

6-20. The Pushlist and the Interpreter are organized such that the last quantity inserted into the Pushlist is normally the first one to be released. Thus, the list has a last-in, first-out behavior. A pushdown list with a last-in, first-out behavior can be used to compute extremely long equations without storing partial results outside the pushdown list. However, it is sometimes more advantageous to calculate partial results which can be used several times. This is not possible with a common pushdown list since a stored quantity is accessible only once. Therefore, the Interpreter provides a modified pushdown list, referred to as Pushlist, which allows repetitive recall of data. If the last-used locations of Pushlist are read out, the second-last-used locations are prepared for read-out as usual. If any other location is read out, the information in that location remains and the "last-used locations" are kept ready for read-out.

6-21. A routine to compute

$$\sin ab + \cos ab + (ab)^2 = x$$

can be written as follows:

```
DMPR        0
            A
            B        → 6
DSQ         1
ROUND
            6        → 8
COS         1
DAD
            6        → 8
```

```
SIN       1
DAD
          6
STORE     X
```

Let us assume that location 6 of the Pushlist is the one to be loaded next. Since no store address is given in the first string, the product ab is stored in location 6 (and 7) of the Pushlist as indicated by the arrow. The second string takes the product from location 6 (and 7), squares it, rounds it, and stores $(ab)^2$ in location 8 (and 9). The third string computes cos ab, adds $(ab)^2$ to it because no second operand address is given, and stores the result again in location 8 because no store address is given. The fourth string computes sin ab, adds cos ab + $(ab)^2$ to it, and stores the final result in location X.

6-22. INDEXING OPERATIONS

6-23. Two index registers per Work Area (table 15-15) are provided for the Interpreter to increase the addressing capability. A ONE in bit position 2 of a Dual-Quantity Instruction Order Code (table 15-13) indicates that the respective Interpretive Instruction has to operate with an indexed address. Bit position 1 of the Interpretive Address Word relevant to the instruction indicates which of the two index registers is to be used. A ZERO indicates Index Register 1; a ONE indicates Index Register 2. The content of the indicated index register is subtracted from the thirteen-bit quantity contained in bit positions 2 through 14 of the relevant address word. After the subtraction, the address of the operand is available. Consider the following string and assume that Index Register 1 contains the quantity - ξ

```
DAD*      0
          α, 1
          β, 1
STORE*    ζ, 1
```

DAD* means instruction code DAD with a ONE in bit position 2. This bit now specifies order code 036 instead of 034 (table 15-13). STORE* represents the STORE code referring to the address to be indexed. When the Interpreter recognizes codes DAD* and STORE*, it indexes addresses α, β, and ζ by the content of Index Register 1 as indicated by the 1 following the comma after α, β, and ζ. The indexed addresses are $\alpha + \xi = A$, $\beta + \xi = B$, and $\zeta + \xi = C$. The sum of the quantities stored at locations A and B is computed and stored at C.

6-24. Consider the following example to demonstrate the usefulness of the indexed string described in the last paragraph. Assume that

sums a + b = c, d + e = f, and g + h = j have to be computed frequently and that the double precision quantities a through j are stored at locations A through J. Some saving in program locations is obtained when the string listed in the preceding paragraph is executed repeatedly and the content of the index register is properly modified at the same time; for instance, from - ξ to - η, and to - ζ. The content of an index register is not changed when a simple string such as the one listed in the preceding paragraph is executed. Modifying the content of an index register is accomplished by executing Index Register Instructions (table 15-13) provided for this purpose. Such a manipulation does not change the content of the multiprecision or vector accumulator nor of any temporary storage location (except the index register involved).

6-25. When the computer is operating in the interpretive mode, index registers can be used also for simple single precision "bookkeeping". For example, if register COUNT is to be incremented by three, a string is written as follows:

LXA 1	1
INCR 1	SXA 1
	COUNT
	3
	COUNT

When the Interpreter recognizes LXA 1 and COUNT (load index register 1 with the content of COUNT), it enters the content of COUNT into Index Register 1. When the Interpreter recognizes INCR 1, 3 it increments the content of Index Register 1 by the quantity 3. The instruction SXA 1, COUNT returns the incremented quantity to location COUNT.

6-26. BINARY POINT

6-27. Normally, the Interpreter assumes that the binary point of any single-precision quantity is placed between bit positions 15 and 14. In other words, the Interpreter assumes that all 14 value bits are placed immediately to the right of the binary point. Therefore, the quantity N contained in a register represents the value N (2^{-14}). For a double-precision quantity, the Interpreter assumes that all 28 value bits are placed immediately to the right of the binary point. For a triple-precision quantity, the Interpreter assumes that all 42 value bits are placed the same way. The weight factors $(2^n, n \lesseqgtr 0)$ give fractional quantities the proper meanings. The weight factors are not stored in the AGC but are built into the programs. An example is given in the following paragraph.

6-28. A program to compute

$$abc + a + b + c = x$$

with weight factor 2^{18} for quantity A, 2^{6} for B, and 2^{3} for C can be written as follows:

```
DMPR        1
DMPR
            A
            B
            C
TSRT        3
DAD         TSRT
DAD         TSRT
DAD         ROUND
            C
            3
            B
            12
            A
            9
STORE       X
```

First the rounded product abc is computed and stored in the Pushlist. The first TSRT instruction then enters quantity c into the MPAC and shifts quantity c three places to the right. The shift places quantity c in the proper weight relation to quantity b, which is then added to quantity c. The second TSRT instruction shifts quantity c + b twelve places to the right to achieve the proper weight relation to quantity a, which is then added to quantity c + b. The third TSRT instruction shifts quantity c + b + a another nine places to the right to establish the proper relation to the product abc. The weight factor of product abc is 2^{27}. The last DAD instruction takes product abc from the Pushlist and adds it to the shifted sum a + b + c. The result is rounded and stored at location X. Using TSLT instead of TSRT operations is impractical because of the overflows which might occur.

6-29. BANK SWITCHING

6-30. There is a restriction concerning program flow and bank switching. A complete address (15-bit address) eliminates most bank switching problems associated with the relevant addresses of Basic Instructions. However, it is still true that program flow must not cross bank boundaries. Control must be transferred with an ITC, for example, to the beginning of the next bank to continue a program once it has reached the end of a bank.

6-31. WORD FORMATS AND CODES

6-32. All words used in the interpretive language fall into two major classes:

Interpretive Instruction Words (IIW's)
Interpretive Address Words (IAW's)

An IIW contains two order codes, or one order code and an integer indicating the number of IIW's immediately following the first IIW of a string (figure 15-12). This number, when incremented by one, indicates the total number of instruction words contained in an instruction string. An IAW might contain an E address, a complete higher FS address, various codes, or a quantity to be used for indexing or shifting.

6-33. TRUE AND FIXED FORM

6-34. Interpretive Instruction Words and Interpretive Address Words (except an Inactive Address) are stored in F memory however, they are not in their true form but in a form which makes decoding easier for the Interpreter. Therefore, a true word, as used in the previous examples must be distinguished from the fixed word, as stored in Fixed Memory. A fixed Interpretive Instruction Word is derived by incrementing (by one) and complementing a true IIW which always contains a ZERO in bit position 15. Consequently, a fixed IIW always contains a ONE in bit position 15. The fixed words are derived from the true words by the Yul Programming System (Issue 13). The Interpreter recomplements a fixed IIW and decrements it by one thus, the true IIW is made available.

6-35. A true IAW representing a complete FS address always contains a ONE in bit position 15 because interpretive program portions operate only in the higher banks (paragraph 6-6). A fixed IAW is derived from such a true IAW by incrementing the true IAW (by one) and eliminating the ONE in bit position 15. Whenever the Interpreter recognizes that is is dealing with an FS address, it decrements the fixed IAW and adds a ONE into bit position 15 to make the true IAW available.

6-36. A true IAW which does not represent a complete FS address or an Inactive Address but represents an E address, a quantity, or a code

is transformed into a fixed IAW by incrementing only. The Interpreter decrements the fixed IAW to make the ture IAW available. A true IAW as well as a fixed IAW may contain a ONE in bit position 15 however, in most cases, it contains a ZERO. An IAW which is the first (or only) word of an address string must always contain a ZERO in bit position 15 to indicate the beginning of the address string.

6-37. The true form and the fixed form of an Inactive Address (paragraph 6-18) are identical and do not need any transformation. In the following paragraphs, reference is made normally to the true form of any IIW or IAW.

6-38. INTERPRETIVE ADDRESS WORDS AND STORE CODES

6-39. The basic format of an Interpretive Address Word (true form) is shown in figure 6-1. Fifteen bits are available for addressing a location in the AGC (table 15-3). The following rules to be observed when addressing is done by an IAW. As usual, all addresses are given in octal numbers.

a. Addresses 00000 through 00052 refer to locations in a Work Area (each consisting of 43 locations, table 15-15) rather than to a CP register or a counter. The initial address of the Work Area in use is provided by the Job Control (Executive) and is stored in location WORKLOC. By adding an address from 00000 through 00052 to the address stored in WORKLOC, the real address of a location in the appropriate Work Area is obtained.

b. Addresses 00053 through 00057 must not be used in an Interpretive Program.

c. Addresses 0060 through 01776 refer to general erasable storage as usual. The use of address 01777 (last in E memory) is illegal. (Incrementing 01777 results in 02000, a code which would confuse the Interpreter.)

d. Complete addresses 42000 through 71776 refer to higher FS storage. Addressing any lower FS storage or the last location of any bank (43777, 45777, etc.,) is illegal.

e. An IAW which contains a quantity between 32000 and 37776 represents a Store Code Address Word composed of a STORE code and a ten-bit address specifying a location in E memory.

6-40. If an IAW contains ONE's in bit positions 11, 13, and 14, and ZERO's in bit positions 12 and 15, then the IAW contains the STORE code (figure 6-1; the use of this STORE code is discussed in paragraphs 6-10 and 6-12). An IAW containing this STORE code refers directly to a location in general erasable memory or to a location in a Work Area. In either case the location is defined by the address contained in bit positions 1 through 10.

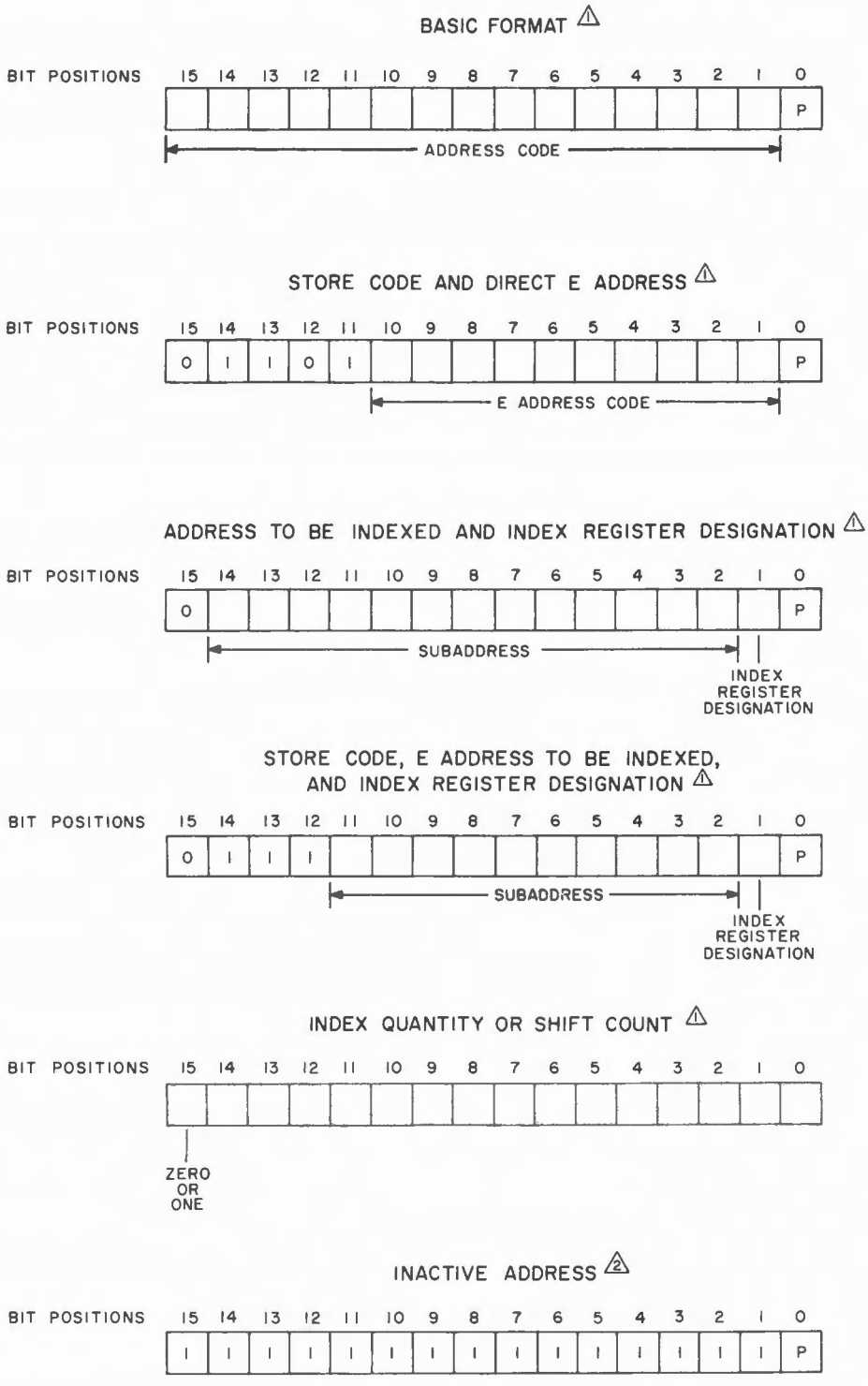
6-41. Many Interpretive Instructions can refer to one of two index registers to modify a given address (paragraph 6-23). Bit position 1 of the relevant IAW (true form) defines which of the two index registers is to be used, and bit positions 2 through 14 contain a 13-bit subaddress (figure 6-1). The content of the specified index register is subtracted from the 13-bit subaddress and thus provides an indexed address. This address can be used in the normal fashion to address a certain memory location. An index register may contain any quantity between +37777 and -37777. This allows subaddress indexing by any binary number with a ONE in bit position 14 to generate an address larger than 17777.

6-42. An IAW with a STORE address can refer also to an indexed address. The STORE code in this case consists of three ONE's contained in bit positions 12 through 14 and a ZERO contained in bit position 15 (figure 6-1). Again, the index register designation is contained in bit position 1 of the IAW. An IAW with the index STORE code always refers to a location in E memory and is derived by indexing the subaddress contained in bit positions 2 through 11. Any indexed STORE address between 0000 and 0052 refers to a Work Area as described in paragraph 6-35.

6-43. INTERPRETIVE INSTRUCTION WORDS AND ORDER CODES

6-44. The various classes of Interpretive Instructions are discussed in paragraphs 15-90 through 15-97, and the order codes are listed in table 15-13. The various order code formats (true forms) are shown in figure 6-2. A Dual-Quantity Instruction code contains ZERO's in bit positions 1 and 2 if it refers to a direct address. A Dual Quantity Instruction code contains a ONE in bit position 2 and a ZERO in bit position 1 if it refers to an indexed address. Whenever the Interpreter recognizes these two prefix bits, it automatically indexes the subaddress contained in the IAW as described in paragraph 6-41.

6-45. Either the respective direct address or the indexed address can be used with no special precaution if: (a) a Dual-Quantity Instruction does not have to load an accumulator before calling the second



⚠ TRUE FORM ONLY

⚠2 TRUE AND FIXED FORM

Figure 6-1. Interpretive Address Word Formats

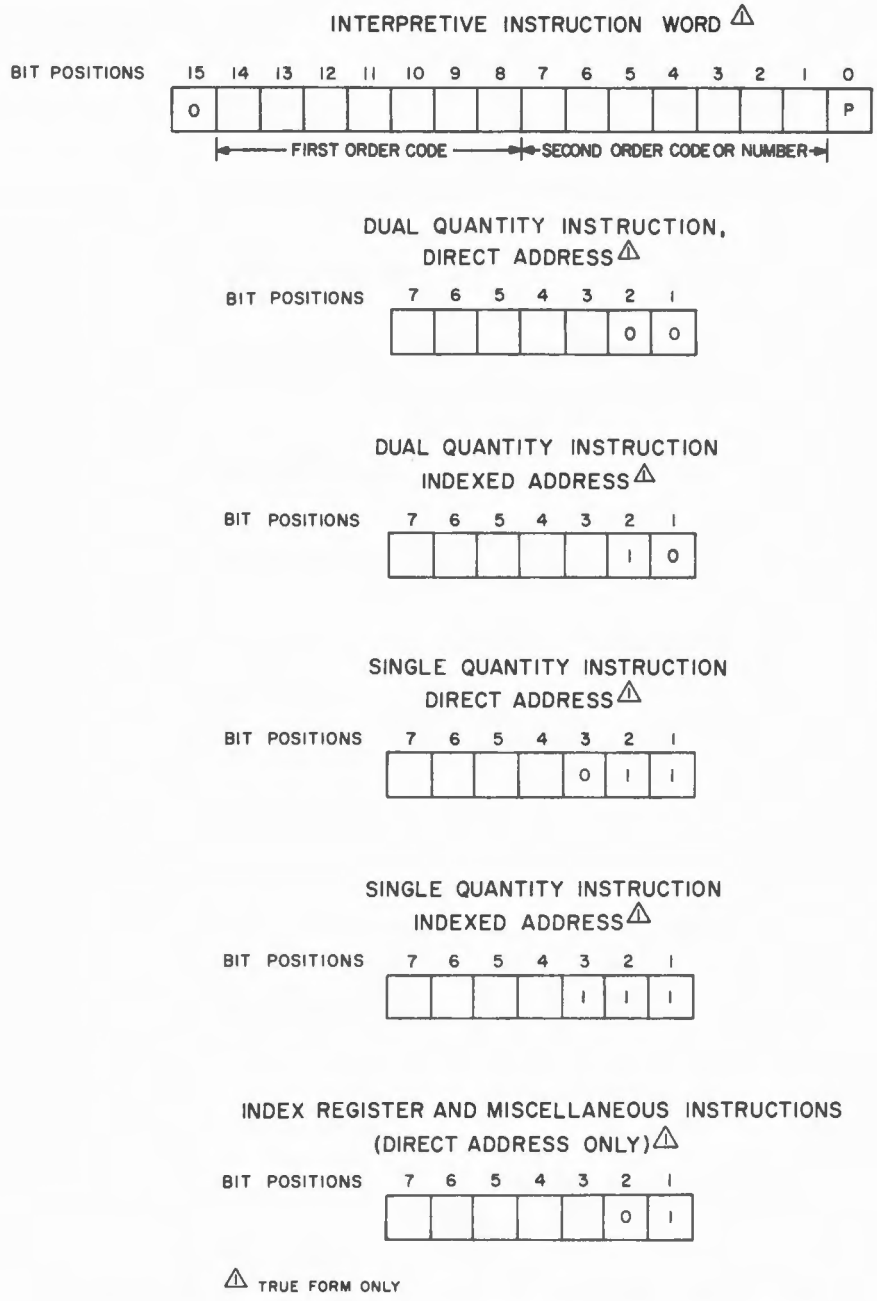


Figure 6-2. Interpretive Instruction Word and Order Code Formats

quantity (second instruction of paragraph 6-10) or (b) both the load address and the operand address are either direct or indexed. If the two addresses are not of the same type, the accumulator must be loaded by executing instruction DMOVE, TMOVE, or VMOVE.

6-46. Single-Quantity Instructions operate with one quantity, normally contained in the MPAC or the VAC. Their order codes can be recognized by the two ONE's in bit positions 1 and 2. The Single- Quantity Instructions are able to enter a quantity into an accumulator if the order code is the first one of an instruction string (paragraph 6-13). For this reason, a third prefix bit has been added to distinguish between direct addressing and indexed addressing. Bit position 3 of the order code contains a ONE when indexing is required, which is then carried out in a way similar to that mentioned in paragraph 6-41.

6-47. Index Register Instructions and Miscellaneous Instructions do not enter a quantity into MPAC or VAC, and refer to direct addresses only. Their order codes carry a ONE in bit position 1 and a ZERO in position 2.

6-48. The 71 Interpretive Instructions provided for the AGC are defined in table 6-1. Common properties of the various instructions are indicated by notes (1) through (7) which are explained at the end of the table.

6-49. Table 6-2 lists the order code bit configuration for each type of instruction. The order code bit configuration is derived from table 15-13.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS

Initials and Name	Definition
DAD K DP Add △1 △2	Adds the double precision (DP) quantity located at K to the DP quantity contained in the MPAC. Keeps the result in the MPAC. Sets register OVFIND in case of overflow or underflow. b(MPAC, MPAC + 1) + c(K, K + 1) = c(MPAC, MPAC + 1)
TAD K TP Add △1 △3	Adds the triple precision (TP) quantity located at K to the TP quantity contained in the MPAC. Keeps the result in the MPAC. Sets register OVFIND in case of overflow or underflow. b(MPAC, MPAC+1, MPAC+2) + c(K, K+1, K+2) = c(MPAC, MPAC+1, MPAC+2)
DSU K DP Sub- tract △1 △2	Subtracts the DP quantity located at K from the DP quantity contained in the MPAC. Keeps the result in the MPAC. Sets register OVFIND in case of overflow or underflow. b(MPAC, MPAC + 1) - c(K, K + 1) = c(MPAC, MPAC + 1)
TSU K TP Sub- tract △1 △3	Subtracts the TP quantity located at K from the TP quantity contained in the MPAC. Keeps the result in the MPAC. Sets register OVFIND in case of overflow or underflow. b(MPAC, MPAC+1, MPAC+2) - c(K, K+1, K+2) = c(MPAC, MPAC+1, MPAC+2)
BDSU K DP Subtract Backwards △1 △2	Subtracts the DP quantity contained in MPAC from the quantity located at K. Keeps the result in the MPAC. Sets register OVFIND in case of overflow or underflow. c(K, K + 1) -b(MPAC, MPAC + 1) = c(MPAC, MPAC + 1)

CONFIDENTIAL

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS(cont)

Initials and Name	Definition
DMP K DP Multiply △1 △2	Multiplies the DF quantity contained in the MPAC by the quantity located at K. Keeps the TP result in the MPAC. Normally, only a DP result is transferred to storage. However, the TP result is transferred to storage if the last instruction of an instruction string is a TP instruction. b(MPAC, MPAC + 1) · c(K, K + 1) = c(MPAC, MFAC + 1, MPAC + 2)
DMPR K DF Multiply and Round △1 △2	Multiplies the DP quantity contained in the MPAC by the quantity located at K. Rounds the result to a DP quantity and keeps it in the MPAC. b(MPAC, MPAC + 1) · c(K, K + 1) = c(MPAC, MPAC + 1) rounded, c(MPAC + 2) = 0
DDV K DP Divide △1 △2	Divides the quantity in the MPAC by the quantity of K if the absolute value of the quantity located at K is larger than the absolute value of the quantity contained in the MPAC. Keeps the DP quotient in the MPAC and turns off the overflow indicator. b(MPAC, MPAC + 1) ÷ c(K, K + 1) = c(MPAC, MPAC + 1) The Interpreter sets register OVFIND and exits the current subprogram, however, the content of the MPAC may be unchanged if the absolute value of the quantity located at K is equal to or smaller than the absolute value of the quantity contained in the MPAC.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
BDDV K DP Divide Backwards 1 2	Divides the quantity at K by the quantity in the MPAC if the absolute value of the quantity contained in the MPAC is smaller than the absolute value of the quantity located at K. Keeps the DP quotient in the MPAC and resets register OVFIND. c(K, K + 1) ÷ b(MPAC, MPAC + 1) = c(MPAC, MPAC + 1) Similar actions are taken as described for the second and third possibilities of DDV K if the absolute value of the quantity contained in the MPAC is equal to or larger than the absolute value of the quantity located at K.
TSRT N TP Shift Right 1 2	Shifts the content of the MPAC N places to the right ($0 \leq N \leq 42$ decimal), where N is a number written into the program instead of an address. Keeps the shifted content in the MPAC. Normally, a DP quantity is transferred to storage after a TSRT operation. If a TP result is to be stored, the end of an instruction string must be a TP instruction. Similarly, if a TP quantity is to be entered into the MPAC for shifting, the instruction string has to start with a TMOVE instruction.
TSLT N TP Shift Left 1 2	Instruction is similar to TSRT with the exception that the content of the MPAC is shifted N places to the left. Sets register OVFIND if any ONE of a positive quantity or any ZERO of a negative quantity is shifted out of the MPAC. The content of the MPAC is shifted N places to the right if the number N is indexed and if N is a negative number ($-42 \leq N \leq -1$ decimal for TP, $-28 \leq N \leq -1$ for DP of vector operations).

CONFIDENTIAL

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
TSLC K TP Shift Left and Count 1 2	Shifts the content of the MPAC (and MPAC+1 and MPAC+2) to the left until a ONE within a positive quantity or a ZERO within of a negative quantity is moved into bit position 14 of the MPAC. Stores the complement of the number of single-position shifts carried out at K. If MPAC contains a plus or minus zero, a plus zero is stored at K.
SIGN K Set Sign into MPAC 1 2	K must be a location in E memory. If c(K) ≥ 0, no operation. If c(K)< 0, the content of MPAC (if in DP or TP mode) or content of VAC (if VEC mode) is complemented. If c(K) = 0, leave sign in MPAC unchanged.
BPL K Branch on Plus 1 4	The instruction located at K is executed next if the MPAC contains a positive quantity. The consecutive instruction is executed normally if the MPAC contains another quantity. The content of the MPAC is not stored in the Pushlist, and execution of the next string is started immediately if there are no more instructions in the current string to be executed and no STORE address is given.
BZE K Branch on Zero 1 4	The instructions located at K are executed next if the MPAC contains a plus or minus zero. All definitive statements for BPL K except the first apply to BZE K.
BMN K Branch on Minus 1 4	The instructions located at K are executed next if the MPAC contains a negative quantity. All definitive statements for BPL K except the first apply to BMN K.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
BHIZ K Branch on High Order Zero △1 △4	The instructions located at K are executed next if the MPAC (not necessarily MPAC + 1 and MPAC + 2) contains a plus or minus zero. All definitive statements for BPL K except the first apply to BHIZ K.
VAD K Vector Add △1 △5	Adds the vector stored at K to the vector contained in the VAC. Keeps the result in the VAC. Sets register OVFIND in case of any overflow or underflow. b(VAC, . . . , VAC + 5) + c(K, . . . , K + 5) = c(VAC, . . . , VAC + 5)
VSU K Vector Subtract △1 △5	Subtracts the vector stored at K from the vector contained in the VAC. Keeps the result in the VAC. Sets register OVFIND in case of any overflow or underflow. b(VAC, . . . , VAC + 5) - c(K, . . . , K + 5) = c(VAC, . . . , VAC + 5)
BVSU K Vector Subtract Backwards △1 △5	Subtracts the vector contained in the VAC from the vector stored at K. Keeps the result in the VAC. Sets register OVFIND in case of any overflow or underflow. c(K, . . . , K + 5) - b(VAC, . . . , VAC + 5) = c(VAC, . . . , VAC + 5)

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
VXSC K Vector times Scalar △1	The Interpreter clears the VAC and enters the vector stored at the first address of the address string into the VAC if the instruction code is in the first of an instruction string. The vector contained in the VAC is then multiplied by the DP scalar stored at the second address of the string. Each vector component is rounded to DP. The result is kept in the VAC. The vector contained in the VAC is multiplied by the DP scalar stored at K if the instruction code is not the first one of an instruction string and if the Interpreter is in the vector mode. Each vector component is rounded to DP. The result is kept in the VAC. $b(VAC, \ldots, VAC + 5) \cdot c(K, K + 1) = c(VAC, \ldots, VAC + 5)$ rounded The DP scalar contained in the MPAC is multiplied by the vector stored at K if the instruction code is not the first one of the instruction string and if the Interpreter is in the DP mode. Each vector component is rounded to DP. The result is kept in the VAC. $b(MPAC, MPAC + 1) \cdot c(K, \ldots, K + 5) = c(VAC, \ldots, VAC + 5)$ The Interpreter transfers the result into storage if the instruction code is the last one of an instruction string. Storage is at the last address of the address string or in the Pushlist when no address is given.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
Dot K Vector Dot Product △1	The Interpreter clears the VAC and enters the vector stored at the first address of the address string into the VAC if the instruction code is the first one of an instruction string. The inner product (vector dot product) is then computed from the vector contained in the VAC and the vector stored at the second address of the address string. The resulting TP scalar is kept in the MPAC. Sets register OVFIND in case of overflow or underflow. The inner product is calculated from the vector contained in the VAC and the vector stored at K if the instruction code is not the first of an instruction string. The resulting TP scalar is kept in the MPAC. Sets register OVFIND in case of overflow or underflow. b(VAC, . . . , VAC + 5) DOT c(K, . . . , K + 5) = c(MPAC, MPAC + 1, MPAC + 2) Normally, only a DP scalar is transferred to storage. However, the TP scalar is transferred to storage if the last of an instruction string is a TP instruction. The Interpreter transfers the DP scalar to storage if the instruction code is the last one of an instruction string. This storage is at the last address of the address string or in the Pushlist when no address is given.
VPROJ K Vector Project △1 △5	Computes the inner product (see DOT K) from the vector contained in the VAC and the vector stored at K. Multiplies the vector contained in the VAC by the scalar of the inner product. Keeps the result in the VAC. Sets register OVFIND in case of overflow or underflow. [b(VAC, . . . , VAC + 5) DOT c(K, . . . , K + 5)] b(VAC, . . . , VAC + 5) = c(VAC, . . . , VAC + 5)

CONFIDENTIAL

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
VXV K Vector Cross Product △1 △5	Computes the outer product (vector cross product) from the vector contained in the VAC and the vector stored at K. Keeps the result in the VAC. Sets register OVFIND in case of any overflow. b(VAC, . . . , VAC + 5) CROSS c(K, . . . , K + 5) = c(VAC, . . . , VAC + 5)
MXV K Matrix times Vector △1 △5	Computes the premultiplication of the vector $\overline{V}_b$ contained in the VAC by the DP matrix M stored at K ($M \cdot \overline{V}_b = \overline{V}_c$). Keeps the result, V_c, in the VAC. Sets register OVFIND in case of any overflow. c(K, . . . , K + 18) · b(VAC, . . . , VAC + 5) = c(VAC, . . . , VAC + 5)
VXM K Vector times Matrix △1 △5	Computes the post multiplication of the transposed vector $\overline{V}_b'$ of the vector $\overline{V}_b$ contained in the VAC by the transposed DP matrix M stored at K ($\overline{V}_b' \cdot M' = \overline{V}_c'$). Keeps the result, transposed vector $\overline{V}_c$, in the VAC. Sets register OVFIND in case of any overflow or underflow. b(VAC, . . . , VAC + 5) · c(K, . . . , K + 18) = c(VAC, . . . , VAC + 5)
VSRT N Vector Shift Right △1 △5	Shifts each vector component of the vector contained in the VAC N places to the right ($0 \leq N \leq 28$ decimal). The quantity N is a number written into the program instead of an address. Each vector component is rounded to DP. Keeps the shifted and rounded vector quantities in the VAC.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS(cont)

Initials and Name	Definition
VSLT N Vector Shift Left 1 5	Shifts each vector component of the vector contained in the VAC N places to the left ($0 \leq N \leq 28$ decimal). The quantity N is a number written into the program instead of an address. Keeps the shifted vector quantities in the VAC. Sets register OVFIND if a ONE of a positive quantity or a ZERO of a negative quantity is shifted out of VAC, VAC+2, or VAC+4. VSLT N with N = 1 must be used to double the vector contained in the VAC. Instruction VAD VAC does not work because the content of VAC can not be added to the content of VAC. Instruction VXSC is not useful either, because a scalar can never be equal to or larger than one. The content of the MPAC is shifted N places to the right if the number N is indexed and if N is a negative number ($-42 \leq N \leq -1$ decimal for TP, $-28 \leq N \leq -1$ for DP of vector operations).
ITC K Interpretive Transfer Control 1 6	Instruction ITC K must be the last (or only) instruction of an instruction string. It causes the instructions stored at K to be executed next. The instruction stores the beginning Interpretive address of the next instruction string at QPRET. The quantity K must be an address in F memory where Interpretive Programs are stored.
STZ K Store Zero 1 6	The instruction stores a single-precision zero in K. The content of the MPAC or the VAC is transferred to storage if the instruction code is the last one of an instruction string and if no STORE address is given. Starts executing the next string of instructions without transferring the MPAC or VAC contents into the Pushlist if the instruction code is the last one of an instruction string and if no STORE address is given.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
BOV K Branch on Overflow (1) (6)	Resets register OVFIND if it is on and executes the instructions located at K; otherwise the execution of the current instruction string is continued. Overflow or underflow during the following operations turns on the overflow indicator: DAD, TAD, DSU, TSU, BDSU, DDV, BDDV, TSLT, VAD, VSU, BVSU, DOT, VPROJ, VXU, MXV, VXM, and VSLT. The MPAC or the VAC content is transferred to storage if the instruction code is the last one of an instruction string but only if a STORE address is given. The Interpreter starts executing the next instruction string without transferring the MPAC or the VAC content into the pushdown list.
SIN DP Sine Function (2)	Computes 1/2 sin $2\pi a$, where a is the before (prior) DP content of the MPAC. Keeps the DP result in the MPAC. The values lie between +1/2 and -1/2 (+1 and -1 scaled). $1/2 \sin [2\pi b\ (MPAC, MPAC + 1)] = c(MPAC, MPAC + 1)$
COS DP Cosine Function (2)	Computes 1/2 cos $2\pi a$, where a is the prior DP content of the MPAC. Keeps the DP result in the MPAC. The values lie between -1/2 and +1/2 (-1 and +1 scaled). $1/2 \cos [2\pi b\ (MPAC, MPAC + 1)] = c(MPAC, MPAC + 1)$
ASIN DP Arcsine Function (2)	Computes $1/2\ \pi$ arcsin y, where y is the prior DP content of the MPAC. Keeps the DP result in the MPAC. The principal values lie between -1/4 and +1/4 ($-\pi/2$ and $+\pi/2$ scaled). $1/2\ \pi \arcsin [2b(MPAC, MPAC + 1)] = c(MPAC, MPAC + 1)$

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
ACOS DP Arccos Function △2	Computes 1/2 π arccos y where y is the prior DP content of the MPAC. Keeps the DP result in the MPAC. The principal values lie between 0 and 1/2 (0 and π scaled). 1/2π arccos [2b(MPAC, MPAC + 1)] = c(MPAC, MPAC + 1)
DSQ DP Square △2	Squares the prior DP content of the MPAC. Keeps the TP result in the MPAC. Normally only a DP result is transferred to storage. However, the TP result is transferred to storage if the last of an instruction string is a TP instruction. square [b(MPAC, MPAC + 1)] = c(MPAC, MPAC + 1)
SQRT DP Square Root △2	Obtains the square root of the prior content (DP or TP) of the MPAC. Keeps a DP result in the MPAC. A negative result is an error and a diagnostic routing is automatically executed. square root [b(MPAC, MPAC + 1, MPAC + 2)] = c(MPAC, MPAC + 1)
ABS Absolute Value △2	Computes the absolute value of the prior content (DP or TP) of the MPAC. Keeps the (DP or TP) result in it. Normally only a DP result is transferred to storage. However, the TP result is transferred to storage if the last of an instruction string is a TP instruction. b(MPAC, MPAC + 1, MPAC + 2) = c(MPAC, MPAC + 1, MPAC + 2)

CONFIDENTIAL

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
DMOVE DP Move	This instruction must be the first of an instruction string. It enters the DP quantity located at the first address K of the address string into the MPAC. c(K, K + 1) = c(MPAC, MPAC + 1), c(MPAC + 2) = 0
TMOVE TP Move	This instruction must be the first of an instruction string. It enters the TP quantity located at the first address of the address string into the MPAC. c(K, K + 1, K + 2) = c(MPAC, MPAC + 1, MPAC + 2)
TP Declare TP	This instruction can not be the first of an instruction string. It causes the Interpreter to switch to TP mode.
VSQ Vector Square △1	The Interpreter clears the VAC and enters the vector stored at the first address of the address string into the VAC if the instruction code is the first one of an instruction string. The vector contained in the VAC is then multiplied (inner product) by the same vector. The resulting TP scalar is kept in the MPAC. Sets register OVFIND in case of overflow or underflow. The vector contained in the VAC is multiplied (inner product) by the same vector if the instruction code is not the first one of an instruction string. The resulting TP scaler is kept in the MPAC. Sets register OVFIND in case of overflow or underflow. b(VAC, . . . , VAC + 5) DOT b(VAC, . . . , VAC + 5) = c(MPAC, MPAC + 1, MPAC + 2)

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
	Normally only a DP scalar is transferred to storage. However, the TP scalar is transferred to storage if the last of an instruction string is a TP instruction. The Interpreter transfers the DP scalar to storage if the instruction code is the last one of an instruction string. Storage is at the last address of the address string or in the Pushlist when no address is given.
ABVAL Vector Absolute Value △5	Computes the half length of the vector $\overline{V}_b$ contained in the VAC and keeps the result $\frac{\overline{V}_b}{2}$ in the MPAC. Normally only a DP result is transferred to storage. However, the TP result is transferred to storage if the last instruction of an instruction string is a TP instruction. Instruction ABVAL stores the square length of vector $\overline{V}_b$ at locations 00034 and 00035.
UNIT Normalize Vector △5	Computes the half-unit vector of the vector $\overline{V}_b$ contained in the VAC and keeps the normalized vector $\frac{\overline{V}_b}{2V_b}$ in the VAC. The magnitude of vector $\overline{V}_b$ must be greater than 2^{-21} because of squaring effect. Instruction UNIT stores the square length of vector $\frac{\overline{V}_b}{2}$ at locations 00034 and 00035, and the length of the vector $\frac{\overline{V}_b}{2}$ at locations 00036 and 00037. Location 00033 is used for temporary storage.
VMOVE Vector Move	This instruction must be the first one of an instruction string. It enters the vector located at K into the VAC. $c(K, \ldots, K+5) = c(VAC, \ldots, VAC+5)$

CONFIDENTIAL

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
VDEF Vector Define △5	Enters a vector, $V = (V_1, V_2, V_3)$, into the VAC. V_1 is taken from the MPAC (and MPAC + 1), V_2 from the top of the Pushlist, and V_3 from just below V_2 in the Pushlist. This instruction can be used to generate a vector whose components must be individually computed in DP and can be stored consecutively. For instance, a program to compute $\overline{V} = (V_1, V_2, V_3)$ with $V_1 = \cos A$, $V_2 = 2B$ and $V_3 = C^2$ is written as follows: DSQ O C TSLT O B 1 COS 1 VDEF A STORE X The three vector components are stored at location X by one vector storing operation instead of by three DP storing operations, thus saving two words of program. (The vector components are entered into the VAC prior to storage at X).
COMP Complement	The Interpreter clears the MPAC and enters the DP quantity located at the first address of the address string into the MPAC if the instruction code is the first one of an instruction string. First VMOVE must be applied to enter a TP quantity into the MPAC or to enter a vector into the VAC before using complementing instruction TMOVE. (continued on next page)

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
	COMP complements the MPAC content and keeps the complemented quantity if the Interpreter is in the DP or TP mode. COMP complements the VAC content and keeps the complemented vector if the Interpreter is in the vector mode.
SMOVE Single Precision Move	This instruction must be the first one of an instruction string. It enters the single precision quantity located at the first address K of the address string into the MPAC. $c(K) = c(MPAC),\ c(MPAC + 1,\ MPAC + 2) = 0$ This instruction is useful for picking up single precision quantities left by input output routines and changing them to double precision.
AXT T, N Address to Index True 	Enters the quantity N into the specified index register ($-37777 \leq N \leq +37776$). The quantity N is written into an address string instead of a relevant address K.
AXC T, N Address to Index Complement 	Enters the complemented form of the quantity N into the specified index register ($-37777 \leq N \leq +37776$). The quantity N is written into an address string instead of a relevant address K.
INCR T, N Increment Index Register 	Increments the content of the specified index register by quantity N ($-37777 \leq N \leq +37776$). The quantity N is written into an address string instead of a relevant address K.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
LXA T, K Load Index Direct from Address ⚠	Enters the SP quantity located at K of E memory into the specified index register. Instruction AXT can be used to load from F memory.
LXC T, K Load Index Comple-mented from Address ⚠	Complements the SP quantity located at K of E memory and enters the complemented quantity into the specified index register.
SXA T, K Store Indexed Quantity in Address ⚠	Transfers the content of the specified index register to location K of E memory.
XCHX T, K Index Regis-ter Exchange ⚠	Exchanges the content of the specified index register with the content of location K of E memory.
XAD T, K Add to Index Register ⚠	Adds the content of location K in E memory to content of the specified index register. Sets register OVFIND in case of overflow or underflow.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
XSU T, K Subtract from Index Register △7	Subtracts the content of location K in E memory from the content of the specified index register. Sets register OVFIND in case of overflow or underflow.
AST T, N Address to Step True △7	Enters the quantity N into the specified step register ($-37777 \leq N \leq +37776$). The quantity N is written into an address string instead of a relevant address K.
TIX T, K Transfer on Index △7	Subtraction is performed and the instructions located at K are executed next if the content of step register T can be subtracted from the indicator index register T without driving the content of the index register to zero or negative. The content of the index register is left unchanged and no branching takes place if the content of the indicator index register is equal to or smaller than the content of the indicated step register.
EXIT Leave Interpretive Mode △7	This instruction must be the last or only instruction of an instruction string. Causes the Interpreter to complete its turn immediately and switch the AGC back to executing Basic Instructions in the basic mode.
RTB K Return to Basic at K △7	This instruction need not be the last of an instruction string. Causes the Interpreter to complete its turn after the execution of instructions located at K and then switch the AGC back to the basic mode. A basic subroutine sequence of Basic Instructions can be called without leaving the Interpreter by using the RTB instruction.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
NOLOD No Load	Resets register LOADIND. This instruction can be used when the MPAC or VAC contains the desired quantity after executioning the last instruction of the previous instruction string.
LODON Load Indicator On	Sets register LOADIND. This instruction can be used to cause the next instruction to load the MPAC or VAC.
ROUND Round to DP	Rounds the MPAC content to DP and turns on the overflow indicator in case of overflow. c(MPAC + 2) = 0
ITCQ Interpretive Transfer Control to Address Stored in QPRET △	Causes the next instruction located at the address contained in QPRET to be executed.
ITA K Interpretive Transfer of Address △	Transfers the content of QPRET to location K in E memory.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

Initials and Name	Definition
ITCI K Interpretive Transfer Control Indirect (7)	Causes the instructions at the address contained in K to be executed next. Instructions ITA K and ITCI K enable the Interpreter to save, and later use, the return address when a called subroutine performs any ITC K instruction.
SWITCH N (7)	Complements content of switch N, $1 \leq N \leq 15$ $17 \leq N \leq 31$ $33 \leq N \leq 47$ decimal
SWITCH N, K (7)	Executes next instruction at K if switch N is set to ZERO or executes consecutive instruction if switch N is set to ONE if address word contains $1 \leq N \leq 15$ $17 \leq N \leq 31$ $33 \leq N \leq 47$ decimal

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

NOTES:

(1) Address K is a direct address or an indexed address as defined by the order code. Number N is a direct number or an indexed number as defined by the order code.

(2) The Interpreter clears the MPAC and enters the DP quantity located at the first address of the address string into the MPAC if the instruction code is the first one of an instruction string.
The Interpreter transfers the DP result to storage if the instruction code is the last one of an instruction string. Storage is at the last address of address string or in the Pushlist when no address is given.

(3) The Interpreter clears the MPAC and enters the TP quantity located at the first address of the address string into the MPAC if the instruction code is the first one of an instruction string.
The Interpreter transfers the TP result to storage if the instruction code is the last one of an instruction string. Storage is at the last address of address string or in the Pushlist when no address is given.

(4) The Interpreter clears the MPAC and enters the DP quantity located at the first address of the address string into the MPAC if the instruction code is the first one of an instruction string.
The Interpreter transfers the DP quantity contained in the MPAC to storage if the instruction code is the last one of an instruction string but only if a STORE address is given.

(5) The Interpreter clears the VAC and enters the vector stored at the first address of the address string into the VAC if the instruction code is the first one of an instruction string.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (cont)

NOTES:
(cont)

The Interpreter transfers the result to storage if the instruction code is the last one of an instruction string. Storage is at the last address of the address string or in Pushlist when no address is given.

⑥ The Interpreter does not clear or enter anything in the MPAC or VAC if the instruction code is the first one of an instruction string.

⑦ Address K (if given) must be a direct address.
The instruction does not enter anything into any accumulator, does not transfer anything to storage, and does not change the current mode of operation (DP, TP or vector operation). Thus, it may be placed anywhere in an instruction string.
The content of the accumulator in use is not transferred to the Pushlist if the instruction code is the last one of an instruction string and if no STORE address is given.

TABLE 6-2

ORDER CODES

Instructions	Order Code Bits						
	7	6	5	4	3	2 (1)	1
Dual-Quantity Instructions							
ITC K	0	0	0	0	1	x	0
VXSC K	0	0	0	1	0	x	0
VSU K	0	0	0	1	1	x	0
BMN K (2)	0	0	1	0	0	x	0
STZ K	0	0	1	0	1	x	0
BOV K	0	0	1	1	0	x	0
DAD K	0	0	1	1	1	x	0
BHIZ K (2)	0	1	0	0	0	x	0
DSU K	0	1	0	0	1	x	0
DBSU K	0	1	0	1	0	x	0
DMP K	0	1	0	1	1	x	0
TSLT N (2)	0	1	1	0	0	x	0
DDV K	0	1	1	0	1	x	0
BDDV K	0	1	1	1	0	x	0
TAD K	0	1	1	1	1	x	0
TSLC K (2)	1	0	0	0	0	x	0
TSRT N	1	0	0	0	1	x	0
DMPR K	1	0	0	1	0	x	0
TSU K	1	0	0	1	1	x	0
SIGN K (2)	1	0	1	0	0	x	0
MXV K	1	0	1	0	1	x	0
VXM K	1	0	1	1	0	x	0
VAD K	1	0	1	1	1	x	0
BZE K (2)	1	1	0	0	0	x	0
BVSU K	1	1	0	0	1	x	0
VSRT N	1	1	0	1	0	x	0
VSLT N	1	1	0	1	1	x	0
BPL K (2)	1	1	1	0	0	x	0
DOT K	1	1	1	0	1	x	0
VXV K	1	1	1	1	0	x	0
VPROJ K	1	1	1	1	1	x	0

(1) x = 0 for direct addressing
x = 1 for indirect addressing

(2) Group B Dual-Quantity Instruction

TABLE 6-2

ORDER CODES (cont)

Instructions	Order Code Bits						
	7	6	5	4	3	2 (1)	1
Single-Quantity Instructions							
TMOVE	0	0	0	0	x	1	1
VMOVE	0	0	0	1	x	1	1
UNIT	0	0	1	0	x	1	1
ABVAL	0	0	1	1	x	1	1
VSQ	0	1	0	0	x	1	1
ABS	0	1	0	1	x	1	1
ASIN	0	1	1	0	x	1	1
ACOS	0	1	1	1	x	1	1
SIN	1	0	0	0	x	1	1
COS	1	0	0	1	x	1	1
SQRT	1	0	1	0	x	1	1
DSQ	1	0	1	1	x	1	1
COMP	1	1	0	0	x	1	1
DMOVE	1	1	0	1	x	1	1
SMOVE	1	1	1	0	x	1	1
VDEF	1	1	1	1	x	1	1
Index Register Instructions and Miscellaneous Instructions							
EXIT	0	0	0	0	0	0	1
RTB K	0	0	0	0	1	0	1
AXT, T N	0	0	0	1	x	0	1
LXA, T K	0	0	1	0	x	0	1
LXC, T K	0	0	1	1	x	0	1
SXA, T K	0	1	0	0	x	0	1
XCHX, T K	0	1	0	1	x	0	1
INCR, T N	0	1	1	0	x	0	1
XAD, T K	0	1	1	1	x	0	1
XSU, T K	1	0	0	0	x	0	1
AST, T N	1	0	0	1	x	0	1
AXC, T N	1	0	1	0	x	0	1
TIX, T K	1	0	1	1	x	0	1
NOLOAD	1	1	0	0	0	0	1
ROUND	1	1	0	0	1	0	1
ITA K	1	1	0	1	0	0	1
ITCI	1	1	0	1	1	0	1
ON N or OFF N	1	1	1	0	0	0	1
BSON N, K, or BSOFF	1	1	1	0	1	0	1
LODON	1	1	1	1	0	0	1
ITCQ	1	1	1	1	1	0	1

(1) x = 0 for direct addressing
x = 1 for indirect addressing.

6-50. INTERPRETER

6-51. GENERAL DESCRIPTION

6-52. Program section Interpreter consists of two major routines: the Dispatcher (locations 4000 through 4472, 4721 through 4727, and 4746 through 4764) and the Executer (locations 4473 through 4720, 4730 through 4745, 4765 through 5652, and 03, 6000 through 03, 7435). The Dispatcher breaks Interpretive Instruction Words (IIW) into Interpretive Instruction order codes, decodes Interpretive Instruction order codes and Interpretive Address Words (IAW), mates the proper relevant addresses with the instruction codes, enters pertinent information into certain storage registers (table 15-15), and, finally, transfers control to the proper subroutines in the Executer. The Executer consists of a group of subroutines which execute each Interpretive Instruction. The Dispatcher and Executer enter a quantity into MPAC or VAC whenever initial loading is required.

6-53. DISPATCHER

6-54. INTERPRETER ENTRIES

6-55. Figure 6-3 illustrates the functional flow of the Dispatcher and some routines of the Executer which are shown in dotted lines. A detailed flow chart for the same routines is shown in attachment 6-1. There are three different entries into the Dispatcher which are:

a. INTPRET for processing the first IIW of the first string of an interpretive program portion,

b. NEWSTRNG for processing the first IIW of any other string,

c. NEWORDER for processing any other IIW or for transferring a partial or final result to the Pushlist or to another E memory location.

6-56. The Interpreter is entered by executing the Basic Instruction TC INTPRET, which is the last instruction of a series of Basic Instructions preceding an interpretive program portion. Executing TC INTPRET switches the operation of the AGC from the basic mode to the interpretive mode. Whenever the Dispatcher is entered at location

INTPRET, the 12-bit address of the location of instruction TC INTPRET is stored in register AWLOC, and the bank number of the program portion currently executed is stored in register BANKSET. Whenever the Dispatcher is entered at location NEWSTRNG (which is really a re-entry following the execution of a string), the bank number of the interpretive program portion currently executed is re-entered into register BNK.

6-57. After entering the Dispatcher at minor routine INTPRET or re-entering at minor routine NEWSTRNG, and setting registers BNK and BANKSET, the quantity 00001 is entered into LOADIND to indicate that the multiprecision accumulator (MPAC) or the vector accumulator (VAC) has to be loaded before an Interpretive Instruction is executed. Then, the address of the location where the first IIW of a string is situated is stored in register IWLOC. The true form of the first IIW is stored in register ORDER and the address of the last IIW of the same string is computed and entered into register AWLOC. The quantity 00000 is entered into the accumulator A indicating no second order code need be decoded for the first IIW; control is then transferred to minor routine IPROC2. Routine IPROC2 transfers the quantity 00000 to register ORDER and moves the order code contained in the first IIW into bit positions 7 through 1 of the accumulator A.

6-58. Whenever the Dispatcher is entered at minor routine NEWORDER (normally a re-entry following the execution of an Interpretive Instruction or the storage of a result), the content of register NEWJOB is tested to determine whether or not control must return to the Job Control (Executive) to change Jobs. (The purpose of the automatic Job breakpoint is discussed in paragraph 12-13.) If the execution of the current string can be continued, the bank number of the current string is re-entered into register BNK.

6-59. After entry at NEW ORDER and the setting of register BNK, the content of register ORDER is tested to determine whether or not a second order code (contained in bit positions 7 through 1, figure 6-2) of an IIW remain to be decoded. Whenever ORDER contains 00000, indicating no second order code need be decoded, the word following the IIW last decoded is tested to determine whether the following word is another IIW or an IAW. If the next word is an IAW (the first IAW of that string) this indicates that all IIW of that string have been processed; in this case, the result has to be stored in the Pushlist or in another E memory location as described in paragraphs 6-102 through 6-107. If the next word is another IIW, this IIW is stored in its true form in register ORDER and its address is stored in register IWLOC. The second order code of this IIW or 00000 is entered into accumulator A and control is

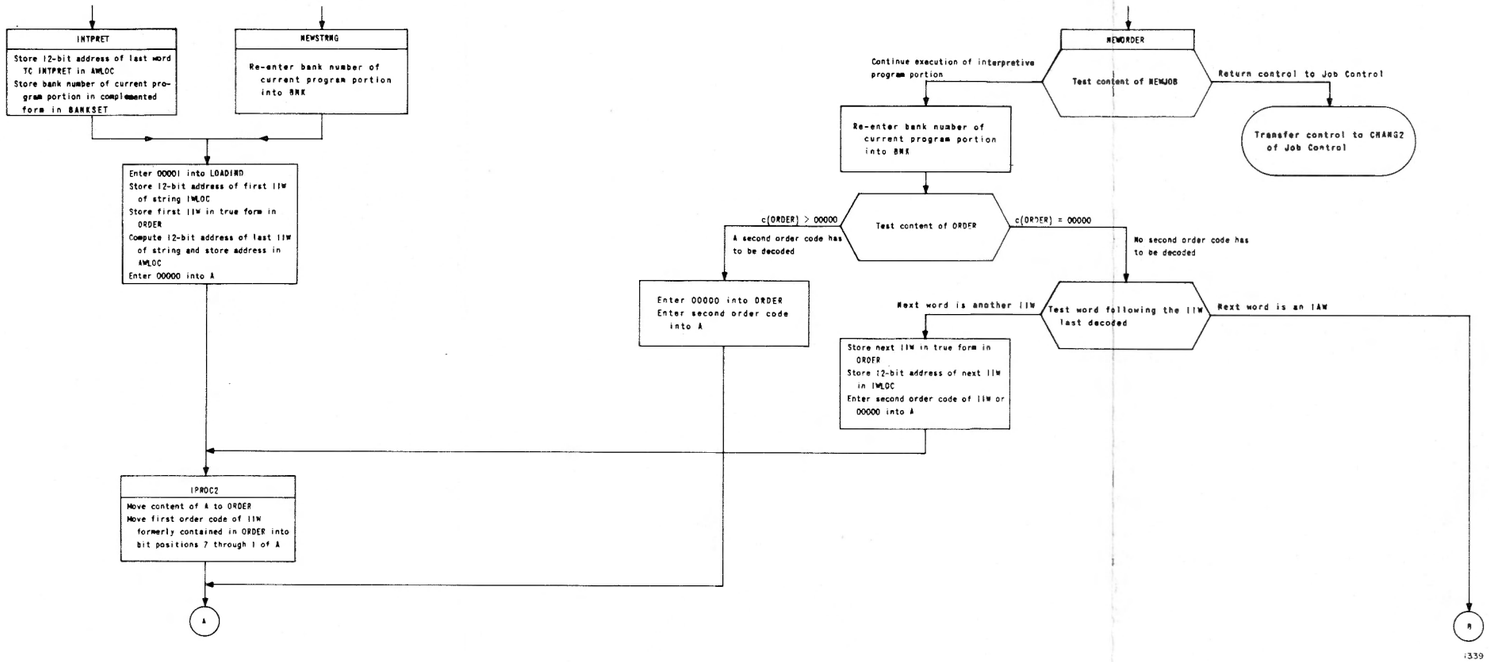


Figure 6-3. Dispatcher Routine Functional Flowchart (Sheet 1 of 3)

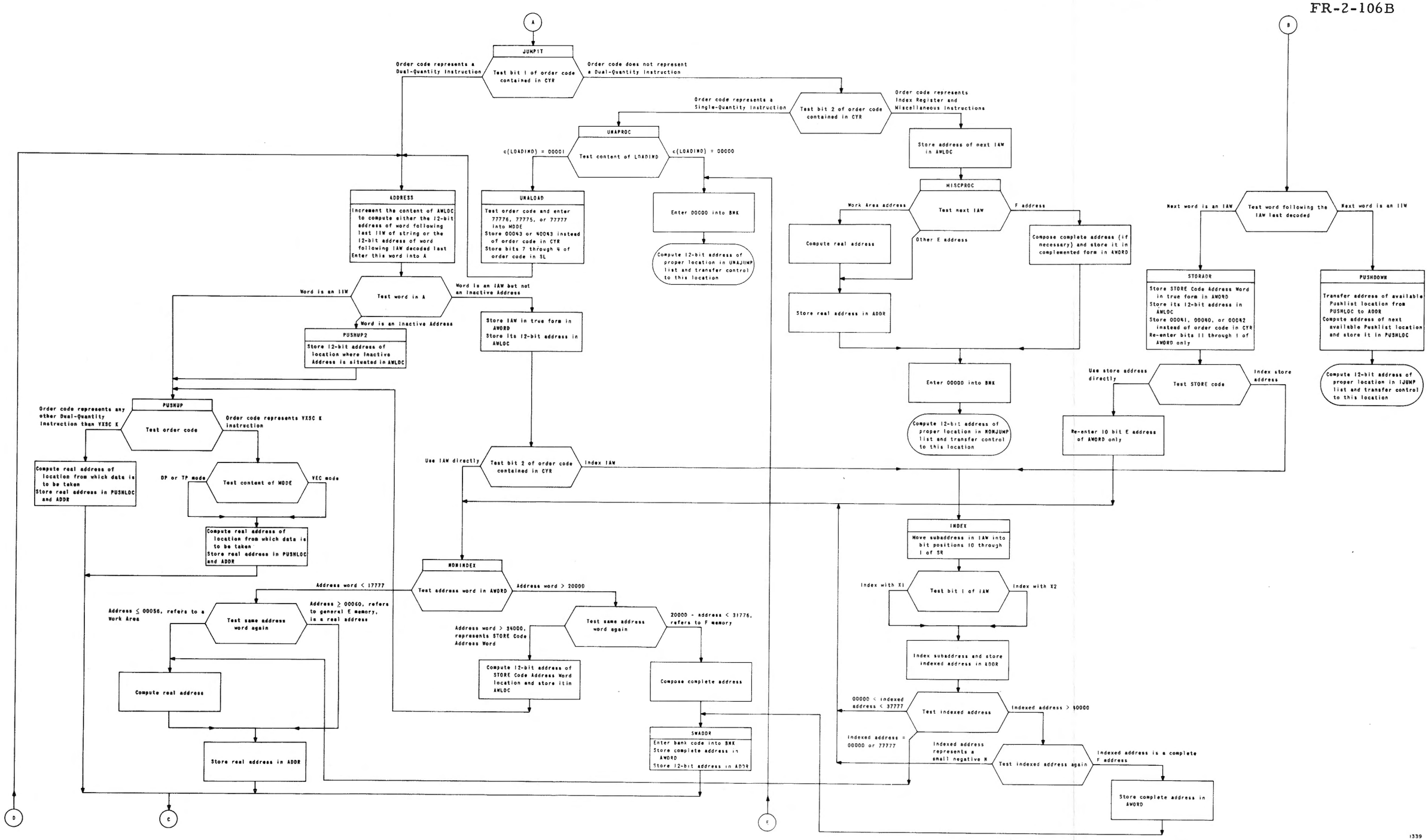


Figure 6-3. Dispatcher Routine Functional Flowchart (Sheet 2 of 3)

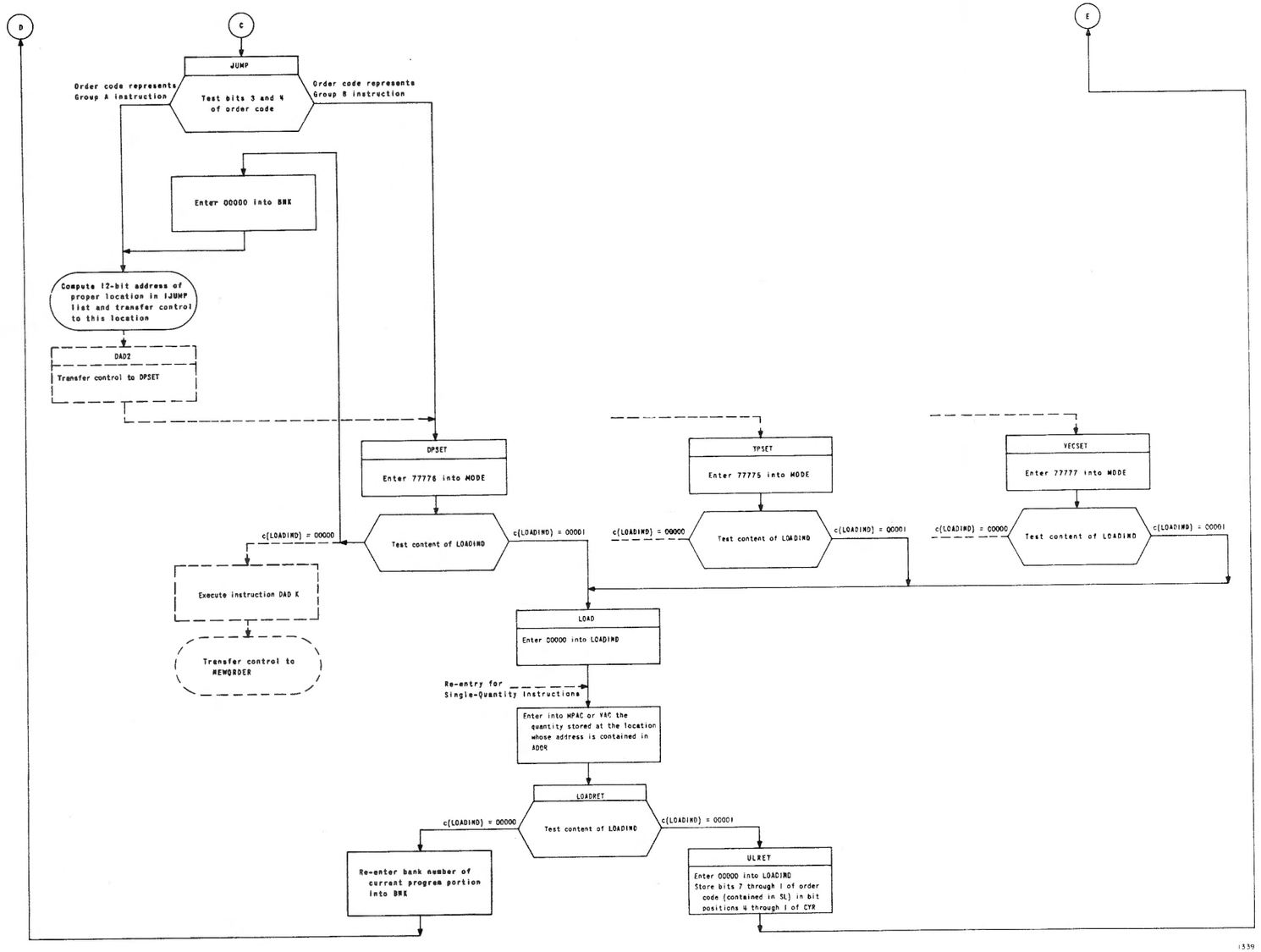


Figure 6-3. Dispatcher Routine Functional Flowchart (Sheet 3 of 3)

transferred to minor routine IPROC2. Routine IPROC2 takes the IIW from register ORDER and enters 00000 or the content of accumulator A in its place. Routine IPROC2 also moves the first order code of the IIW which was previously contained in register ORDER, into bit positions 7 through 1 of the accumulator.

6-60. Whenever register ORDER contains an order code upon reentry into the Dispatcher at location NEWORDER, register ORDER is set to 00000 indicating the decoding of the second order code will be done within a few μsec, and the second order code is already entered into accumulator A for decoding. Following the execution of IPROC2 (point Ⓐ of figure 6-3 or attachment 6-1), accumulator A always contains an order code in true form which has to be decoded.

6-61. EXECUTION OF DUAL-QUANTITY INSTRUCTIONS

6-62. Dual-Quantity Instructions operate with two quantities. Normally, the first quantity is contained in the multiprecision or vector accumulator and the second quantity is contained at a location which is specified by an IAW. If a Dual-Quantity Instruction is the first one of a string, two IAW's are needed to specify where the two operands are stored. The IAW's might be used directly or might be indexed as defined by the order code. When no IAW is given, the Interpreter automatically refers to the Pushlist, but the Interpreter might also refer to the Pushlist if an Inactive Address is given instead of another IAW. Dual-Quantity Instructions can be divided into either Group A or Group B instructions as noted in table 6-2. Dual-Quantity Instructions can also be divided into instructions referring to a relevant address K or instructions using an integer N as noted in table 15-13. First the execution of Dual-Quantity Instructions referring to nonindexed IAW's is described; thereafter, the execution of Dual-Quantity Instructions referring to indexed IAW's is described. Finally, the execution of Dual-Quantity Instructions referring to a Pushlist location is described. Reference is made to the Pushlist either when no IAW or an Inactive Address is given.

6-63. Instructions Referring to Nonindexed IAW's

6-64. Partial Decoding and Mating with an IAW

6-65. When entering minor routine JUMPIT, the order code contained in accumulator A is tested to determine whether it represents (a) a Dual-Quantity Instruction, (b) a Single-Quantity Instruction, or (c) an Index Register Instruction or a Miscellaneous Instruction. Whenever the order code represents a Dual-Quantity Instruction (ZERO in

bit position 1), program control is transferred to minor routine ADDRESS. When a reference is made to an IAW for the first time, routine ADDRESS computes the address of the word following the last IAW of the string (i. e. address of the first IAW of that string). When a reference has been made previously to an IAW, routine ADDRESS computes the address of the word following the IAW decoded last (i. e. address of the next IAW of that string or address of the first IIW of the next string). In either case, routine ADDRESS enters the word into accumulator A.

6-66. If the following word is an IAW which is not an Inactive Address (figure 6-1), the true form of the IAW is stored in register AWORD and its location address is stored in register AWLOC. Next, the order code is tested again to determine whether the order code represents a Dual-Quantity Instruction specifying an IAW to be used directly or an IAW to be indexed (bit 2, a ZERO or a ONE respectively). In the first case, program control is transferred to minor routine NONINDEX. In the second case, control is transferred to minor routine INDEX which is discussed in paragraph 6-82.

6-67. Minor routine NONINDEX tests the address word (true IAW or indexed address) contained in register AWORD to determine whether or not it refers to E memory (address < 2000). If the address word refers to E memory, a differentiation must be made between addresses from 00000 to 00052_8 and those from 00060 to 01776. Any address between 00000 and 00052_8 refers to a location in one of the five Work Areas. If such an address is given, the real address within a Work Area must be computed by adding the given address to the address of the Work Area in use. If an address between 00060 and 01776 is given, this is a real E address. (Addresses 00053 through 00057 and 01777 are illegal, paragraph 6-39.) Either of the real addresses is stored in register ADDR before program control is transferred to minor routine JUMP.

6-68. If the first test of routine NONINDEX indicates that the address does not refer to E memory, the address is tested once more to determine if it refers to F memory (02000_8 < address word < 31776_8) or if it represents a STORE Code Address Word (34000_8 < address word). If the address refers to F memory, a ONE is added into bit position 15 of the IAW to compose the complete address of an F memory location (compare with paragraph 6-35). Finally, minor routine SWADDR enters the bank code of the F address into register BNK; stores the complete address in register AWORD; stores the corresponding 12-bit address into register ADDR; and transfers program control to minor routine JUMP. If the address word tested represents

a STORE Code Address Word, its location address is computed and stored in register AWLOC. Program control is then transferred to minor routine PUSHUP, which is discussed in paragraphs 6-88 and 6-89.

6-69. Following the execution of routine NONINDEX and routine SWADDR when required (point Ⓒ of figure 6-3 or attachment 6-1), register ADDR contains a real 12-bit address within a Work Area, E memory or F memory. Minor routine JUMP does the final decoding of the order code.

6-70. Final Decoding and Execution of Group A Instructions

6-71. When entering minor routine JUMP, the order code is tested the third time. If bit positions 3 and 4 contain a ONE, indicating that the order code represents a Group A Dual-Quantity Instruction, program control is transferred to a location in the IJUMP list. If bit positions 3 and 4 both contain a ZERO, indicating that the order code represents a Group B Dual-Quantity Instruction (table 6-2), program control is transferred to minor routine DPSET.

6-72. Bits 7 through 3 of the order code for a Group A Dual-Quantity Instruction are used to compute the address of the proper location in the IJUMP list. The IJUMP list is a jump list for Dual-Quantity Instructions, store operations, and load operations. The list consists of 35 TC instructions which transfer control to the proper routines in the Executer for executing a particular interpretive operation. Thirty-one TC instructions are provided for Dual-Quantity Instructions, three for transfer operations, and one for load operations. In the case of a Group A instruction, control is always transferred to a routine in FF memory thereby leaving the content of register BNK unchanged.

6-73. Assuming the Dispatcher had to decode the Order code 034, which represents instruction DAD K referring to a direct address (table 15-13). Since bit positions 5 through 3 contain ONE's (table 6-2), the quantity 00007 is added to address 4367 (the last address prior to the IJUMP lists, see program listing of SUNRSE33) and control is transferred to location 4376 which contains instruction TC DAD2. After control is transferred to minor routine DAD 2 (a part of the Executer), this routine immediately transfers control to minor routine DPSET. When the order code represents a TP or VEC operation, control is transferred to minor routine TPSET or VECSET.

6-74. Minor routine DPSET first enters 77776 into register MODE which switches the operation of the Interpreter to the DP mode. Minor

routine TPSET enters 77775 into register MODE and minor routine VECSET enters 77777. After setting the mode, routine DPSET, TPSET or VECSET tests the contents of register LOADIND. If register LOADIND contains 00000, indicating that the MPAC or the VAC need not be loaded (already contains the proper information), program control is returned to routine DAD2 for the actual execution of the DAD K instruction. The address K of a double precision quantity is contained in register ADDR. After executing DAD K, the sum is contained in MPAC and program control is transferred to location NEWORDER to decode the next order code.

6-75. If register LOADIND contains 00001 when tested, a quantity has to be entered into MPAC (or VAC in case of instruction VAD K) before execution of the instruction. (For instance, the quantity 00001 is entered into register LOADIND whenever the first IIW of a string is executed, except when the IIW contains instruction NOLOD.) If register LOADIND contains 00001, program control is transferred to minor routine LOAD. Routine LOAD enters 00000 into register LOADIND and enters into MPAC or VAC the quantity stored at the location whose address is contained in register ADDR. Minor routine LOADRET determines whether register LOADIND contains 00000, indicating it is dealing with a Dual-Quantity Instruction or contains 00001, indicating it is dealing with a Single-Quantity Instruction. When LOADIN contains 00000, routine LOADRET re-enters the bank number of the interpretive program portion being executed into register BNK, and transfers control to routine ADDRESS to decode the next IAW which is now supplying the address for the second operand. If instruction DAD K is the first one of a string, its final execution takes place after the second pass through routine DPSET.

6-76. The final decoding of all Group A Dual-Quantity Instructions is alike; the execution of instruction DAD K has been discussed as an example. The execution of other Group A Dual-Quantity Instructions will be described with the Executer.

6-77. Final Decoding and Execution of Group B Instructions

6-78. The execution of Group B instructions is very similar to the execution of Group A instructions. For Group A instructions, routine JUMP first transferred control to the IJUMP list and the Executer before testing the content of register LOADIND, loading MPAC or VAC, and supplying a new operand address. For Group B instruction, routine JUMP transfers control immediately to routine DPSET (never to TPSET or VECSET) and tests the content of register LOADIND. If register LOADIND contains 00000, control is returned to routine JUMP

which first enters 00000 into register BNK. Thereafter, it computes the proper location in the IJUMP list and transfers control to this location. Control is then transferred to the proper routines in bank 03 for the execution of the Group B instruction. The quantity 00000 is entered into register BNK because the jump list for Group B instructions only transfers control to Executer routines situated in bank 03. If register LOADIND contains 00001, the MPA or VAC is loaded and a new operand address is supplied before the location in the jump list is computed.

6-79. Decoding and Execution of Instructions TSRT N, TSLT N, VSRT N, and VSLT N.

6-80. Instructions TSRT N, TSLT N, VSRT N and VSLT N do not refer to an address (K) as do all other Dual-Quantity Instructions but refer instead to an integer (N) specifying the number of places a quantity is to be shifted (paragraph 6-28). Whenever the execution of such a shift instruction is requested, the relevant IAW does not contain an address (K) but a small positive number (N). This number is handled by minor routines ADDRESS and NONINDEX, the same way as an address is handled. At point Ⓒ of figure 6-3, register ADDR contains a quantity equal to (N + Work Area address used) where $0 < N < 42$. After entering routine JUMP, the operations described earlier take place. Executer subroutine TRUE2 (not shown on figure) subtracts the Work Area address and makes number N available before the actual execution of one of the four shift instructions.

6-81. Instructions Referring to Indexed IAW's

6-82. The decoding operations described in paragraphs 6-65 and 6-66 also apply for the decoding discussed here. If the second test of the order code discloses that bit position 2 contains a ONE, program control is transferred to minor routine INDEX.

6-83. Minor routine INDEX shifts the IAW one place to the right to position the subaddress (figure 6-1) in bit positions 10 through 1. Then bit 1 of the IAW (true form, last contained in register AWORD) is tested to determine whether this address has to be indexed by the content of index register X1 or X2. Thereafter, the content of the proper index register (X1 or X2) is subtracted from the subaddress and the final product is an indexed address (an indexed shift number for instructions TSRT N, TSLT N, VSRT N, or VSLT N) which is stored in register AWORD.

6-84. If the indexed address is equal to 00000 or 77777, the first register of a Work Area is addressed and program control is transferred directly to the subroutine of routine NONINDEX which computes the real address of the first location in the proper Work Area. Thereafter, the operation continues as previously described. If the indexed address is larger than 00000 but smaller than 37777, program control is transferred to routine NONINDEX which computes the real address (if necessary) and enters this address or shift number N into register ADDR as described in paragraphs 6-67 and 6-80. If the indexed address contains a ONE in bit position 15, it might represent a complete F memory address (K) or a small negative quantity (N) used for indexing. In the first case, the complete address is stored in register AWORD and routine SWADDR operates as described in paragraph 6-68. In the second case, program control is transferred to routine NONINDEX which deals with the shift number (N) the same way as described in paragraph 6-80. At point © of figure 6-3, register ADDR contains an indexed address or a quantity equal to (N + Work Area address used), where $-43 < N < 42$.

6-85. Instructions Referring to Pushlist Locations

6-86. Decoding of Order Codes Which Cannot Be Mated with an IAW

6-87. The decoding operation described in paragraph 6-65 also applies for the decoding discussed here; however, the operation described in paragraph 6-66 does not. If the word following the IAW last used is an IIW (the first IIW of the next string, paragraph 6-65), program control is transferred to minor routine PUSHUP.

6-88. Minor routine PUSHUP tests the order code being decoded to determine if it represents instruction VXSC K or another Dual-Quantity Instruction. If the order code represents any Dual-Quantity Instruction other than VXSC K, routine PUSHUP computes the real address of that location from which data is to be taken. This is accomplished by: subtracting the quantity 00002 from the content of register PUSHLOC if the Interpreter is in the DP mode; subtracting the quantity 0003 if the Interpreter is in the TP mode; or subtracting the quantity 00006 if the Interpreter is in the VEC mode. When the real address has been computed, it is stored in register PUSHLOC and entered also into register ADDR. The real address, referring to a Pushlist location, is then used the same as any other address entered into register ADDR.

6-89. If the first test of routine PUSHUP indicates instruction VXSC K has to be executed, the content of register MODE is tested. If the

Interpreter is in the VEC mode, indicating that a vector has to be multiplied by a scalar, the quantity 0000_2 is subtracted from the content of register PUSHLOC. If the Interpreter is in the DP or TP mode, indicating that a scalar has to be multiplied by a vector, the quantity 00006 is subtracted from the content of register PUSHLOC. Again, a real address, referring to a Pushlist location, is entered into registers ADDR and PUSHLOC and then used as any other real address.

6-90. Decoding of Order Code and Mating with an Inactive Address

6-91. The decoding operation described in paragraph 6-65 also applies for the decoding discussed herein. If the word following the IAW used last is an Inactive Address (figure 6-1), program control is transferred to minor routine $PUSHUP_2$.

6-92. Minor routine $PUSHUP_2$ increments the content of register AWLOC to store in it the address location where the Inactive Address is situated. Thereafter, program control is transferred to routine PUSHUP which operates as described in paragraphs 6-88 and 6-89.

6-93. EXECUTION OF SINGLE QUANTITY INSTRUCTIONS

6-94. When entering minor routine JUMPIT (paragraph 6-65), the order code contained in the accumulator A is tested. Whenever the order code represents a Single-Quantity Instruction (ONE's in bit positions 1 and 2), bit 2 of the order code is tested immediately and program control is transferred to minor routine UNAPROC.

6-95. Minor routine UNAPROC tests the content of register LOADIND to determine whether or not MPAC or VAC has to be loaded. If register LOADIND contains 00000, the quantity 00000 is entered into register BNK; bits 7 through 4 of the order code are used to compute the address of the proper location in the UNAJUMP list (jumplist for Single-Quantity Instructions, 16 TC instructions), and program control is transferred to this location. Control is then transferred to the proper routines in bank 03 for the execution of the instruction. The quantity 00000 is set into register BNK because most TC instructions within the UNAJUMP list transfer control to Executer routines which are situated in bank 03. After executing a Single-Quantity Instruction, program control is transferred to location NEWORDER to decode the next order code.

6-96. If the test of routine UNAPROC discloses that register LOADIND contains 00001, program control is transferred to minor routine UNALOAD. Minor routine UNALOAD tests the order code to determine

whether it represents a TP instruction (TMOVE), a VEC instruction (VMOVE through VSQ of table 6-2), or a DP instruction (ABS through VDEF). For a TP instruction, the quantity 77775 is entered into register MODE; for a VAC instruction, the quantity 77777; and for a DP instruction, the quantity 77776. Then the quantity 43 is stored in bit positions 6 through 1 of register CYR and a ONE is entered into bit position 15 if the order code represents a Single-Quantity Instruction which is also an indexed address. Bits 7 through 4 of the order code are stored in register SL for later use and program control is transferred to routine ADDRESS.

6-97. Minor routine ADDRESS operates as described earlier, thereby making use of the content of register CYR to determine whether or not an address has to be indexed. If indexing is required, minor routine INDEX is executed prior to minor routine NONINDEX; otherwise, NONINDEX is executed immediately. If routine ADDRESS cannot find either another IAW which is then an Inactive Address, or no IAW at all, reference is made to the Pushlist. Regardless of the path taken after leaving routine UNALOAD to reach point © of figure 6-3, register ADDR contains a real address and register CYR contains the information entered by routine UNALOAD.

6-98. After entering routine JUMP, the path for Group A Dual-Quantity Instructions is followed. The quantity 00043 stored in register CYR is used to transfer control to the last location of the IJUMP list. Program control is then transferred to routine LOAD which now enters a DP, a TP, or a VEC quantity (dependent on the content of register MODE) into the MPAC or VAC. Routine LOADRET tests the content of register LOADIND, finds that it contains 00001, indicating a Single-Quantity Instruction, and transfers control to routine ULRET. Minor routine ULRET enters 00000 into register LOADIND, stores bits 7 through 4 of the order code in bit positions 4 through 1 of register CYR, and transfers control to routine UNAPROC. Routine UNAPROC operates as described in paragraph 6-95.

6-99. EXECUTION OF INDEX REGISTER INSTRUCTIONS AND MISCELLANEOUS INSTRUCTIONS

6-100. When entering minor routine JUMPIT (paragraph 6-65), the order code contained in the accumulator is tested. Whenever the order code represents an Index Register Instruction or a Miscellaneous Instruction (a ONE in bit position 1 and a ZERO in bit position 2), bit 2 two of order code is tested immediately, the address of the next IAW is stored in register AWLOC, and program control is transferred to minor routine MISPROC.

6-101. Minor routine MISPROC tests the next IAW (true form complemented) to determine whether it represents (a) an address in a Work Area, (b) another address in E memory, or (c) an address in F memory. For the first two possibilities a real address is entered into register ADDR. Otherwise, the address can only refer to the higher banks of FS memory. If a ONE is not present in bit position 15 of the IAW, a ONE is added to this bit position; the complete address is complemented and stored in register AWORD. In any case, routine MISPROC enters the quantity 00000 into register bank, computes the address of the proper location in the NONJUMP list (jump list of index Register Instructions and Miscellaneous Instructions, 16 TC instructions) and finally transfers program control to this location. Next, control is transferred to the proper routines in bank 03 for the actual execution of the instruction. After the execution of the instruction, program control is transferred to location NEWORDER to decode the next order.

6-102. STORING OF RESULTS

6-103. Storing a Result at a Store Address

6-104. Once all order codes of a string have been decoded and all Interpretive Instructions of that string have been executed, the Dispatcher is re-entered at location NEWORDER. First the content of register NEWJOB is tested and register BNK is set as described in paragraph 6-58. Then the content of register ORDER, which contains 00000, is tested as described in paragraph 6-59. As the word following the IIW decoded last is tested, it is found that the following word is the first IAW of the string. This indicates all Interpretive Instructions of the string have been executed and the Dispatcher now looks for a location to store the information contained in MPAC or VAC. The word following the IAW decoded last is now tested to determine if it is an IAW (STORE CODE Address Word, figure 6-1) or an IIW (first IIW of next string). In the first case, program control is transferred to minor routine STORADR to store the result at a given address; in the second case, control is transferred to minor routine PUSHDOWN to store the result in the Pushlist.

6-105. Minor routine STORADR enters the true form of the STORE Code Address Word into register AWORD and its address location into register AWLOC. Thereafter, the quantity 00041 is stored in register CYR if the Interpreter is in the DP mode, the quantity 00040 if in the TP mode, or the quantity 00042, if in the VEC mode. The content of register AWORD is replaced by bits 11 through 1 of the STORE address word. Next, the STORE Code Address Word is tested to deter-

mine whether or not the STORE code requires that the store address be indexed. A ONE in bit position 12 of the STORE code address indicates indexing is necessary, (figure 6-1). If no indexing of the store address is required (a ZERO in bit position 12), the content of register AWORD is replaced by the 10-bit E address of AWORD and program control is transferred to routine NONINDEX. Routine NONINDEX enters a real address into register ADDR as described earlier.

6-106. Routine JUMP then follows the path for Group A instructions, makes use of the content of register CYR. computes the address of a store transfer instruction in the IJUMP list and transfers control to that location. Places 32 through 34 of the IJUMP list are provided for store operations. Dependent upon which mode the Interpreter is in, control is transferred to one of these three locations. From there, control is transferred to the proper Executer routines. These routines transfer the control of MPAC or VAC to those E registers which are specified by the control of register ADDR. Thereafter, control is transferred to location NEWSTRING for the execution of the next string.

6-107. If indexing of the store address is required, program control is transferred to routine INDEX which operates as described in paragraph 6-83. Routine INDEX enters the indexed store address into register AWORD. An indexed store address can be equal to or larger than 00000 (or 77777) but not larger than 17777. After the execution of routine INDEX, program control is transferred to routine NONINDEX which operates as described in paragraph 6-106.

6-108. Storing a Result in a Pushlist

6-109. The operation described in paragraph 6-104 also applies for the operation discussed here. In addition, minor routine PUSHDOWN transfers the address of the available Pushlist location from register PUSHLOC to register ADDR and enters a new address into register PUSHLOC. The new address is equal to the old address plus 2, 3, or 6, dependent upon whether the Interpreter is operating in the DP, TP, or VEC mode. Finally, the address of place 32, 33, or 34 in the IJUMP list is computed and program control is transferred to this address to initiate the proper store operation. After completion of the store operation, control is transferred to location NEWSTRING.

6-110. EXECUTER

6-111. The description of the Executer operations will be added later.