APOLLO GUIDANCE COMPUTER

Information Series

ISSUE 6

INTERPRETER AND INTERPRETIVE INSTRUCTIONS

FR-2-106A

25 September 1963

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

AGC INFORMATION SERIES

ISSUE 6

INTERPRETER AND INTERPRETIVE INSTRUCTIONS

FR-2-106A

6-1. INTRODUCTION

6-2. This is the sixth issue of the AGCIS published to inform members of the technical staff at MIT and Raytheon about the AGC and the Apollo G & N system. The Interpreter, Interpretive Instructions, and word formats were described briefly in Issue 1. A basic description of the Interpretive Programs and a more detailed description of the operation of the Interpreter are presented in this issue. Information contained in this issue was taken from "A List Processing Interpreter for AGC", AGC Memo No. 2, by C. A. Muntz or received from Mr. Muntz verbally.

6-3. GENERAL

6-4. The Interpreter is a highly sophisticated program consisting of approximately 1500 words stored in Fixed-Fixed (FF) memory and Bank 3 of Fixed-Switchable (FS) memory (table 1-3). The Interpreter executes Interpretive Programs stored elsewhere in the computer. Interpretive Programs are composed of Interpretive Instruction Words and Interpretive Address Words which are arranged in accordance with certain rules. There are 126 different order codes provided for the various Interpretive Instructions (table 1-2). The notation to be used when an Interpretive Program is written is similar to the "parenthesis-free" notation introduced by Lukasiewicz (sometimes referred to as Polish notation). The format or arrangement of an Interpretive Program varies considerably from that of most programs of general-purpose computers. A basic description of Interpretive Programs is presented below prior to a discussion of the Interpreter.

6-5. BASIC DESCRIPTION OF INTERPRETIVE PROGRAMS

6-6. INTERPRETIVE AGC PROGRAMS VERSUS COMMON PROGRAMS

6-7. For many computers the following program can be written for computing a + b = x:

```
CA      A
AD      B
TS      X
```

Instruction CA A (clear and add A) clears the accumulator and enters into it the quantity a stored at location A. Instruction AD B (add B) adds the quantity b located at B to the content of the accumulator. Instruction TS X (transfer to storage X) transfers the content of the accumulator (the sum a + b) to location X. The Basic Programs of the AGC are written in a manner similar to the program illustrated above.

6-8. An Interpretive Program for the AGC to compute a + b = x with double precision can be written as follows:

```
DAD     0
        A
        B
STORE   X
```

This program consists of one Interpretive Instruction Word (IIW) and three Interpretive Address Words (IAW). (For word formats, refer to figure 1-6.) The IIW consists of the order code DAD and a zero. The code DAD represents the instruction double-precision add, which is a Dual-Quantity Instruction (paragraph 1-48). The zero (all ZERO's in bit positions 7 through 1) indicates that no other IIW immediately follows the first (or only) IIW and also that the first IAW immediately follows the first IIW. As the Interpreter (interpreting program of the AGC) decodes the IIW, it automatically clears the MPAC (multiprecision accumulator, consisting of three locations in E memory) and

enters into it the double-precision quantity a stored at locations A (and A+1). Thereafter, the Interpreter causes the double-precision quantity b stored at B (and B+1) to be added to the content of the MPAC. As the Interpreter recognizes the word STORE X, it transfers the content of the MPAC (the sum a + b) to location X (and X+1).

6-9. The designations A, B, and X have no relation to the name of any central register or to special locations in E memory. The expressions (and A+1), (and B+1), and (and X+1) indicate that the double-precision quantities are stored in two memory locations (E or F memory). Reference is normally made only to the first location of a double-precision quantity, of a triple-precision quantity, or of a vector. Double- and triple-precision calculations are carried out with the MPAC (locations MPAC, MPAC+1, and MPAC+2) and vector operation with the VAC (vector accumulator, consisting of six locations in E memory, VAC through VAC+5).

6-10. The string of IIW's in the program of paragraph 6-8 consists of only one word, as indicated by the zero in the first, and only, IIW. The string of IAW's consists of three addresses: a load address, which is always represented by the first IAW of a string; an operand address; and a store address, which is identified by the code STORE.

6-11. A program written for the AGC to compute

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

with double precision can be written 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 program 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 string of IIW's. The Interpreter recognizes also that it has to take the first data from address A; 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 1-2) represents a Single-Quantity Instruction (paragraph 1-49), 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 $\sqrt{a^2 + b}$. The Interpreter next 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: $\cos (b + \sqrt{a^2 + b})$. Now quantity a is 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).

6-12. STORING AND RECALLING OF PARTIAL RESULTS

6-13. When ab + cd = x is computed, it is necessary to store partial results before the final result can be calculated. When

```
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 program is omitted, i.e., when program

```
DMP     0
        A
        B
```

is executed, the Interpreter automatically puts the partial result into the next free location of a pushdown list. Pushdown lists are series of erasable registers reserved for temporary storage.

6-14. One of three methods can be applied to recall a quantity from the pushdown list; these methods are discussed in this and the following two paragraphs. If an instruction finds no address word in the address string, it automatically takes the operand from the proper location of the pushdown list. For example, consider the following program, 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 pushdown list as described in paragraph 6-13. After quantity c is multiplied by quantity d, the Interpreter recognizes that it has to execute an addition. However, the Interpreter cannot find an 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 pushdown list. The final result,

ab + cd = x, is again stored in the pushdown list because no address is given.

6-15. The Interpreter must distinguish between an address in which the result is to be stored (store address) and an address from which data is to be taken (operand address). This is to allow data to be recalled from the pushdown list just before the result is stored elsewhere. Therefore, a special format for a store address which is recognized by the Interpreter is employed. The program

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

is very similar to the program listed in paragraph 6-14, 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 pushdown list and added to the product cd. The final result is written into location X.

6-16. A special code, Inactive Address (77777), references the pushdown list by a certain signal. A program 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 pushdown list, 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 -) tells the computer to take the quantity b^2 from the pushdown list 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. This is the Inactive Address method for recalling a partial result.

6-17. Sometimes more than one location is needed to store partial results. A program 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 program consists of four Interpretive Strings. First d^2 is computed and stored in the pushdown list. Next, c^2 is calculated and kept in the MPAC. After the execution of the first DAD instruction, the quantity $c^2 + d^2$ is stored in the pushdown list. The quantity $a^2 + b^2$ is computed in the same way. After the execution of the second DAD instruction, the quantity $a^2 + b^2$ is contained 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 the pushdown list, divides $a^2 + b^2$ by $c^2 + d^2$, and stores the quotient at X.

6-18. The pushdown list and the Interpreter are organized so that normally the last quantity inserted into the pushdown list is the first one to be released. Thus, the list has a last-in, first-out behavior. With the type of pushdown list described so far, extremely long equations can be computed without partial results stored 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 this pushdown list since a stored quantity is accessible only once. Therefore, the Interpreter provides a modified pushdown list which has repetitive recall of data. If the last-used location of the modified pushdown list is read out, the pushdown list information moves as usual. If any other location is read out, the information in that location remains and the modified pushdown list does not change.

6-19. Assuming that location 6 of the pushdown list is to be loaded next, a program to compute

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

can be written as follows:

```
DMPR        0
            A
            B       →6
NOLOD       1
DSQ       ROUND     →8
COS         1
DAD
            6       →8
SIN         1
DAD
            6
STORE       X
```

Since no store address is given, the product ab is stored in locations 6 and 7 in the pushdown list. Instruction NOLOD keeps the product ab in the MPAC. The product ab is squared, rounded, and stored in locations 8 and 9 of the pushdown list, the next free locations, since no relevant address is given for this instruction string. Instruction COS takes the product ab from locations 6 and 7 in the pushdown list and computes ab. The first DAD instruction takes $(ab)^2$ from locations 8 and 9, adds it to cos ab, and puts cos ab + $(ab)^2$ back into locations 8 and 9. Instruction SIN takes the product ab from locations 6 and 7 and computes sin ab. The second DAD instruction takes cos ab + $(ab)^2$ from locations 8 and 9 and adds it to sin ab. The final result is transferred to location X.

6-20. INDEXING OPERATIONS

6-21. Two index registers per Work Area (table 6-3) are provided for the Interpreter to increase the addressing capability. A ONE in bit position 2 of a Dual-Quantity Instruction Order Code (table 1-2) 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 address contained in bit positions 2 through 14 of the relevant address word. After the subtraction, the final address of the data to be worked with is available. Consider the following program and assume that index register 1 contains the quantity 2:

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

In the program, 1 indicates index register 1. DAD* means instruction code DAD with a ONE in bit position 2. This bit now specifies order code 036

instead of 034 (table 1-2). When the Interpreter recognizes code DAD*, it indexes addresses α and β by the content of the appropriate index register. The indexed addresses are $\alpha - 2 = D$ and $\beta - 2 = E$. Consequently, the sum of the quantities stored at locations D and E is computed and retained in MPAC. In a similar way, the address contained in the Store Code Address Word is indexed and becomes $\gamma - 2 = F$ when the sum is stored.

6-22. The usefulness of indexing an operation is demonstrated by another example. Consider the following locations, A through J, in E memory:

A
B
C
D
E
F
G
H
J

Assume that a + b = c, d + e = f, and g + h = j have to be computed frequently. Some saving in program location is obtained when the program listed in the preceding paragraph is repeatedly executed and the content of the index register is properly incremented at the same time. The content of an index register is not changed when a simple program 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 1-2) provided for this purpose. Such a manipulation does not change the content of an accumulator nor of any temporary storage location. Therefore, the manipulations can be carried out when required by a program.

6-23. When the computer is operating in the interpretive mode, index registers are used also for simple precision "bookkeeping". Such bookkeeping operations are very useful for carrying out repetitive computations such as

those described in paragraph 6-22. For example, if a counter is to be incremented by three and the new quantity is to be used as an address, a program is written as follows:

```
LXA, 1          1
INCR, 1         SXA, 1
                Z
                "3"
                Z
```

When the Interpreter recognizes LXA, 1 and COUNT (load index register 1 with the content of Z), it enters the content of Z into index register 1. When the Interpreter recognizes INCR, 1 and "3", it increments the content of index register 1 by the quantity 3 (not by the quantity located at address 3). The instruction SXA, 1 Z returns the incremented quantity to location Z.

6-24. BINARY POINT

6-25. 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 quantity 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, $n \leq 0$) give the fractional quantities the proper meanings. The weight factors are not stored in the AGC; they are built into the programs. An example is given in the following paragraph.

6-26. 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 pushdown list. 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 pushdown list 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.

6-27. WORD FORMATS AND ADDRESSES

6-28. All words written into an interpretive program (paragraph 1-52) fall into two major classes: Interpretive Instruction Words, which compose instruction strings, and Interpretive Address Words, which compose address strings. Interpretive Instruction Words (figure 1-6) contain two order codes

or one order code and a number indicating the number of Interpretive Instruction Words immediately following the first instruction word. This number, when incremented by one, indicates the total number of instruction words contained in an instruction string. An Interpretive Address Word might contain an address, various codes, or a quantity to be used for indexing or shifting.

6-29. Interpretive Instruction Words and Interpretive Address Words (except Inactive Address) are stored in F memory not in their true form but in a form which makes decoding easier for the Interpreter. For this reason one has to distinguish between a true word (as used in the examples on the previous pages) and the fixed word (the word as it is stored in fixed memory). A fixed Interpretive Instruction Word is derived by incrementing (by one) a true Interpretive Instruction Word (which always contains a ZERO in bit position 15) and complementing. Consequently, a fixed Interpretive Instruction Word always contains a ONE in bit position 15. The Interpreter recomplements a fixed instruction word and decrements it by one. In this way the true Interpretive Instruction Word is made available.

6-30. A fixed Interpretive Address Word is derived by incrementing the true Interpretive Address Word (which normally contains a ZERO in bit position 15) by one. Consequently, a fixed Interpretive Address Word also normally contains a ZERO in bit position 15. A fixed Address Word which is the first (or only) word of an address string must always contain a ZERO in bit position 15. A fixed Address Word which is not the first word of an address string may also contain a ONE in bit position 15. The Interpreter decrements a fixed Address Word and makes the true Interpretive Address Word available. (The true and the fixed forms of the Inactive Address are identical and do not need incrementing or decrementing.) In the following paragraphs, reference is made normally to the true form of any Instruction or Address Word.

6-31. The basic format of an Interpretive Address Word (true form) is shown in figure 6-1. Fourteen bits are available for direct addressing of a location in the AGC (table 1-6). The following rules have to be observed when addressing is done by means of an Interpretive Address Word. All addresses are given in octal numbers as usual.

a. Addresses 00000 through 00052 refer to locations in a Work Area rather than to a flip-flop register or counter. Memory space for five Work Areas (each consisting of 43 locations) is provided to enable the computer to run five programs alternately. The initial address of the Work Area in use is provided by the Executive Program and is stored in location 00105 (WORKLOC). By adding an address from 00000 through 00052 to the address stored in WORKLOC, the address of a location in the appropriate Work Area is obtained.

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

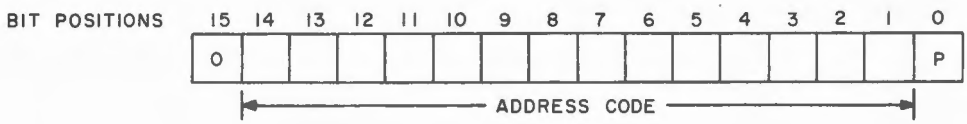
c. Addresses 0060 through 01776 refer to general erasable storage as usual. The use of address 01777 (last of E memory) is illegal. (Incrementing 01777 results in 02000, a code which would confuse the Interpreter.)

d. No reference may be made to any location in FF memory by an Interpretive Address Word except for an Address Word associated with the Interpretive Instruction RTB (return to basic mode).

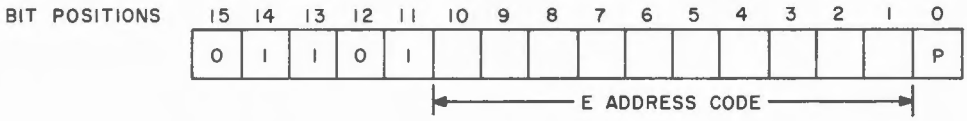
e. Addresses 06000 through 31776 refer to FS storage as usual. The use of address 31777 (last of Bank 14) is illegal.

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

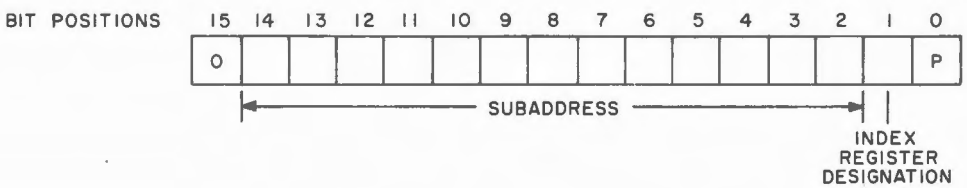
BASIC FORMAT ①



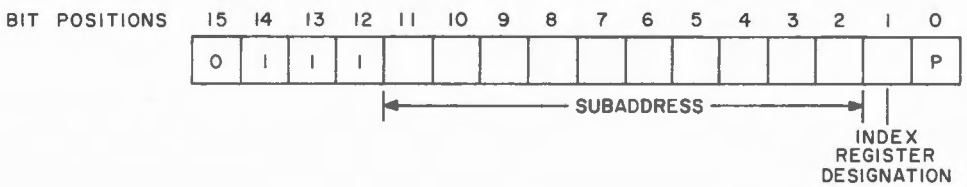
STORE CODE AND DIRECT E ADDRESS ①



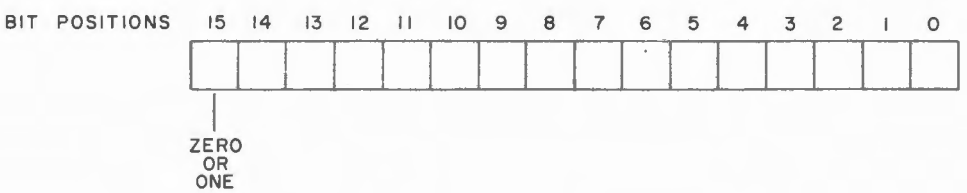
ADDRESS TO BE INDEXED AND INDEX REGISTER DESIGNATION ①



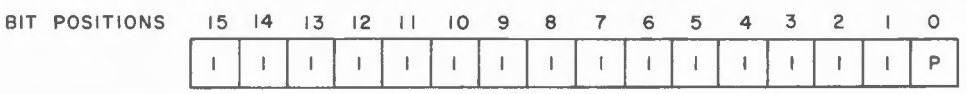
STORE CODE, E ADDRESS TO BE INDEXED,
AND INDEX REGISTER DESIGNATION ①



INDEX QUANTITY OR SHIFT COUNT ①



INACTIVE ADDRESS ②



① TRUE FORM ONLY

② TRUE AND FIXED FORM

Figure 6-1. Interpretive Address Word Formats

g. No reference may be made to any location in Banks 21 through 34 by an Interpretive Address Word except for an Address Word associated with the Interpretive Instruction RTB.

6-32. If an Interpretive Address Word has a ONE in bit positions 11, 13, and 14 and a ZERO in bit positions 12 and 15, then the Address Word contains the code STORE. Code STORE is discussed in paragraph 6-12 and is shown in figure 6-1. An Interpretive Address Word containing the code STORE refers 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.

6-33. Many Interpretive Instructions can refer to one of two index registers to modify a given address (paragraph 6-21). Bit position 1 of the Address Word 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 a true Interpretive 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 in order to generate an Interpretive Address larger than 17777.

6-34. An Interpretive Instruction operating with a STORE address can refer also to the index registers. The STORE code in this case consists of three ONE's contained in bit positions 12 through 14. Again the Index Register designation is contained in bit position 1 of the Address Word. An Interpretive Address referring to a location in E memory is derived by indexing the subaddress contained in bit positions 2 through 11. (This is similar to the way described in paragraph 6-24.) Any indexed STORE address between 0000 and 0052 refers to a Work Area as described in paragraph 6-31.

6-35. An Interpretive Address Word may contain an Index Quantity (positive or negative quantity used for indexing) or a Shift Count (positive quantity indicating the number of positions an information has to be shifted). The fixed forms of Index Quantity and Shift Count are identical to the true words incremented by one, as they are for the address words discussed previously.

6-36. The Inactive Address, which has been discussed adequately in paragraph 6-11, is the only Interpretive Address Word of which the true and fixed forms are identical.

6-37. There is a restriction concerning program flow and bank switching. A 14-bit address eliminates most bank switching problems associated with 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.

6-38. The various classes of Interpretive Instructions are discussed in Issue 1, and the order codes are listed in table 1-2. 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 Address Word, as described previously.

6-39. Either the respective direct address or the indexed address can be used with no special precaution if (a) a Dual-Quantity Instruction need not load an accumulator before calling the second quantity (paragraph 6-8) or (b) both the load address and the operand address are either direct or indexed.

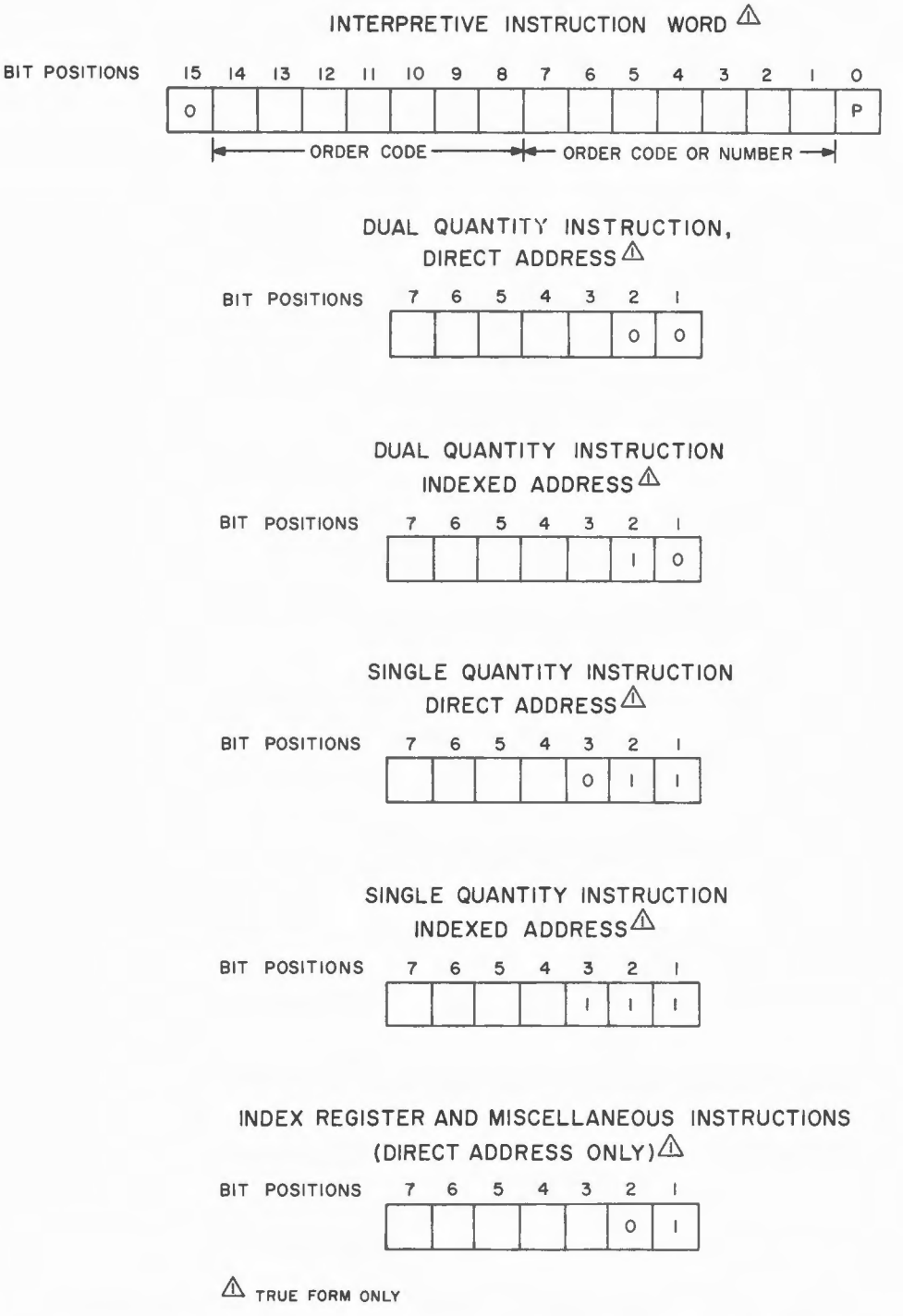


Figure 6-2. Interpretive Instruction Word and Order Code Formats

If the two addresses are not of the same type, the accumulator must be loaded by executing instruction DMOVE, TMOVE, or VMOVE.

6-40. 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 6-8). 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 6-32.

6-41. Because Index Register Instructions and Miscellaneous Instructions do not enter a quantity into MPAC or VAC, they refer to direct addresses only. Their order codes carry a ONE in bit position 1 and a ZERO in position 2.

6-42. INTERPRETIVE INSTRUCTIONS

6-43. The 71 Interpretive Instructions provided for the AGC are defined in table 6-1 (see pages 6-26 through 6-46). Common properties of the various instructions are indicated by (1) through (7) and are explained at the end of the table. Table 6-2 (pages 6-47 and 6-48) indicates the order code bit configuration as derived from table 1-2.

6-44. REGISTER ASSIGNMENTS

6-45. The assignments of E memory registers are listed in table 1-5. The names and purposes for locations 0100 to 0615 which are reserved for the Interpreter, Executive, and Waitlister are given in table 6-3 (pages 6-49 through 6-55).

6-46. INTERPRETER OPERATION

6-47. The Interpreter consists of two major parts, the Dispatcher (314 words in FF memory) and the Executer (616 words in FF memory and 784 words in Bank 3 of FS memory). 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 6-3), and, finally, transfers control to the proper subroutines in the Executer. The Executer consists of a field of subroutines which execute each Interpretive Instruction. The Dispatcher and Executer enter a quantity into MPAC or VAC whenever initial loading is required.

6-48. Three different entries into the Dispatcher are provided as shown in figure 6-3:

INTPRET:	for processing the first IIW of the first string of an Interpretive Program,
NEWSTRNG:	for processing the first IIW of any other string,
NEWORDER:	for processing any other IIW or for transferring a partial (or final result) to E memory or the push-down list.

6-49. Normally, the Interpreter is entered by executing instruction TC INTPRET, which precedes an Interpretive Program and switches the operation of the AGC from the basic mode to the interpretive mode. Whenever the Interpreter is entered at subroutine INTPRET or re-entered at NEWSTRNG after the execution of a string, the number of the bank in which the Interpretive Program is stored is set into register BNK. The quantity 00001 is set into LOADIND to indicate that the MPAC or VAC has to be loaded before

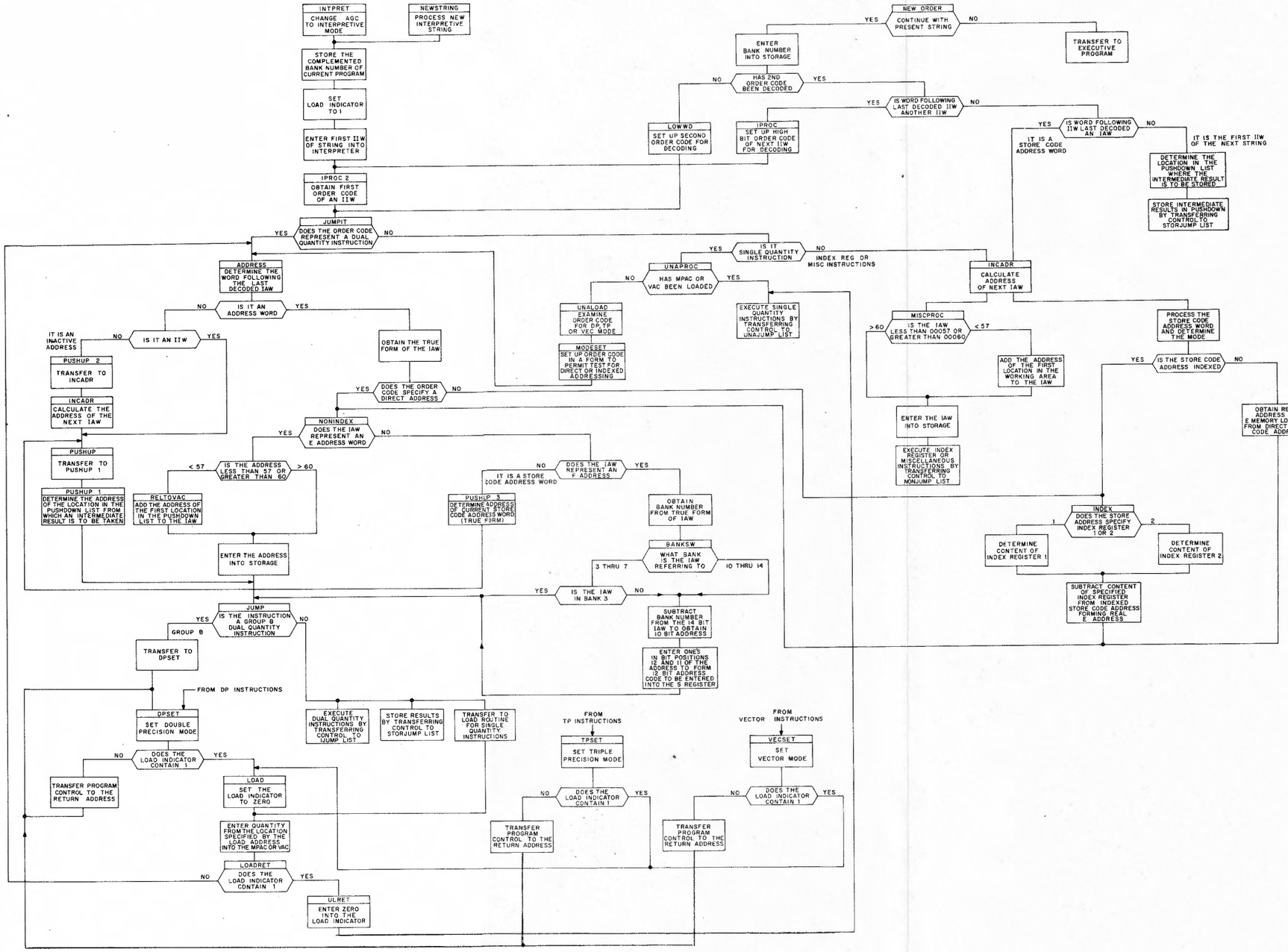


Figure 6-3. Interpreter Flow Chart

an Interpretive Instruction is executed. The order code contained in the first IIW (bits 14 through 8) is obtained (in its true form) for decoding. The quantity 00000 is set into ORDER to indicate that no second order code is contained in the first IIW; i. e. , that no other order code contained in the first IIW has to be decoded. The number (bits 7 through 1) indicating how many IIW's of that string follow the first IIW is used to compute the location of the first IAW of the string.

6-50. If the Interpreter is re-entered at NEWORDER, the Bank number of the program is also set into BNK. The first order code (bits 14 through 8) of the second, third, etc IIW is obtained (in its true form) and the second order code or zero (bits 7 through 1) is stored in ORDER. After the first order code of an IIW is processed, the second order code is processed, and a zero is set into ORDER to indicate that all order codes of that IIW had been obtained for processing. Whenever the Interpreter is re-entered at NEWORDER and it detects an IAW instead of another IIW, this indicates that all IIW's of that string have been processed. Consequently, the Interpreter looks for a location where it can store the partial (or final) result. When a Store Code Address is given, the result will be stored at that address which is contained in the Store Code Address Word. When no Store Code Address is given, the result is stored in the proper location in the proper pushdown list.

6-51. Whenever an order code has been obtained for decoding, it is tested first to determine whether it represents (a) a Dual-Quantity Instruction, (b) a Single-Quantity Instruction, or (c) an Index Register or Miscellaneous Instruction.

6-52. If the order code represents a Dual-Quantity Instruction, the Interpreter looks for an address from which it can take data. When an IAW is given, the order code is tested again to determine whether the IAW has to be

indexed at this time and used from now on instead of the original IAW. Thereafter, the IAW which did not require indexing or the indexed IAW is tested to determine if it represents a location in E or F memory. If the location is in E memory, a differentiation must be made between addresses from 00000 to 00052 and those from 00060 to 01776. Any address between 00000 and 00052 refers to a location in one of the five Work Areas. If this is the case, then the address must have the address of the initial location of the particular Word Area added to it to form a real address. If the address refers to a location in F memory, the Bank number must be subtracted from the IAW and ONE's have to be set in bit positions 12 and 11 to form an address code which can be entered into register S for selecting the proper location for read out.

6-53. If the order code represents a Dual-Quantity Instruction and the Interpreter cannot find an IAW for obtaining data, the Interpreter refers to the pushdown list to obtain a real address (within the pushdown list) from which it can take data.

6-54. Once the location from which data is to be taken has been determined (during the earlier decoding of a Dual-Quantity Order Code), the order code is further examined. When it represents a Group A Dual-Quantity Instruction (first ten instructions of table 1-2) or a Vector Dual-Quantity Instruction, control is transferred to the proper TC instruction in the IJUMP list. This TC instruction transfers control to a proper subroutine in the Executer initiating the actual execution of a particular Dual-Quantity Instruction.

6-55. During the actual execution of a Group A or Vector Dual-Quantity Instruction, first the DP, TP, or Vector mode is set by entering the quantity 77776, 77775, or 77777, respectively, into MODE. Thereafter, the content of LOADIND is tested. When it is 00001 (as during the execution of an

instruction which is the first one of a string), the quantity stored at the locations specified by the first IAW of the string is loaded into MPAC or VAC. The LOADIND is set to 00000, and control is returned to the Dispatcher in order to search for another address from which the second quantity can be taken. Once this address has been determined (similar to the way described in paragraphs 6-52 and 6-65), control is again transferred to the Executer to continue the actual execution of that instruction. Again, the DP, TP, or Vector mode is set. When the sontent of LOADIND is tested, the quantity 00000 becomes apparent, which indicates that loading MPAC or VAC is not necessary. The actual execution of the instruction is continued and completed. Thereafter, the Dispatcher is re-entered at NEWORDER.

6-56. In case the LOADIND contains 00000 during the first test (as is normally the situation during the execution of a Group A or Vector Dual-Quantity Instruction which is not the first one of a string), no loading of MPAC or VAC is necessary. The quantity located at that address which has been specified by the Dispatcher is called, and the actual execution of the particular instruction is carried out by the Executer without interruption. Thereafter, the Dispatcher is re-entered again at NEWORDER.

6-57. When the test of paragraph 6-54 indicates that the order code represents a Group B Dual-Quantity Instruction (BMN K, BHIZ K, TSLT K, TSLC K, SIGN K, BZE K, OR BPL K), the DP or TP mode is set and the MPAC is loaded immediately if LOADIND contains 00001. If LOADIND contains 00000, the DP or TP mode is set and control is transferred to the proper TC instruction in the IJUMP list, as described in paragraph 6-54.

6-58. When the test of paragraph 6-51 indicates that the order code represents a Single-Quantity Instruction, the content of LOADIND is tested immediately. If LOADIND contains 00001, the location from which data is to be taken

(and entered into MPAC or VAC) is determined in the same way as described in paragraphs 6-52 and 6-53. Once the data location has been defined, MPAC or VAC is loaded and LOADIND is set to 00000. Whenever LOADIND contains 00000 (during the test mentioned earlier in this paragraph or after loading), control is transferred to the proper TC instruction in the UNAJUMP list. This TC instruction transfers control to a proper subroutine in the Executer initiating the actual execution of a particular Single-Quantity Instruction.

6-59. When the test of paragraph 6-51 indicates that the order code represents an Index Register or Miscellaneous Instruction, the next IAW is decoded to obtain the address from which the index quantity has to be taken, and the Index Register (X1 or X2 of one of the five Work Areas) to be used for indexing is defined. Thereafter, control is transferred to the proper TC instruction in the NONJUMP list. This TC instruction transfers control to a proper subroutine in the Executer initiating the actual execution of a particular Index Register or Miscellaneous Instruction.

6-60. In paragraph 6-50 it was mentioned that a result is transferred to a Store Address or into a pushdown list. When a Store Code Address Word is given, it is tested to determine whether it has to be indexed or not and if by means of Index Register X1 or X2. Once a real Store Address has been obtained, control is transferred to the proper TC instruction (TC DTS1, TC TTS1, or TC VTS1) in the STORJUMP list. This TC instruction transfers control to the proper subroutine (DP, TP, or Vector transfer to storage) in the Executer initiating the actual transfer of a result. When no Store Code Address Word is given (as indicated by the appearance of the first IIW of the next string instead of a Store Code Address Word during testing of the word following the IAW last decoded), the next free loaction in the proper pushdown list is determined and the result stored there.

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. Turns on the overflow indicator 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. Turns on the overflow indicator 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. Turns on the overflow indicator 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. Turns on the overflow indicator 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 Backwards DP Subtract △1 △2	Subtracts the DP quantity contained in MPAC from the quantity located at K. Keeps the result in the MPAC. Turns on the overflow indicator in case of overflow or underflow. c(K, K + 1) -b(MPAC, MPAC + 1) = c(MPAC, MPAC + 1)

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

Initials and Name	Definition
DMP K DP Multiply 1 2	Multiplies the DP 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, MPAC + 1, MPAC + 2)
DMPR K DP 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 turns on the overflow indicator and exits the current subprogram, not necessarily leaving the content of the MPAC 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 (continued)

Initials and Name	Definition
BDDV K Backwards DP Divide (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 turns off the overflow indicator. 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. Turns on overflow indicator 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).

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

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 of a positive quantity or a ZERO 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, sets sign in MPAC to plus. If c(K) < 0, sets sign in MPAC to minus. If c(K) = 0, leaves 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 pushdown list, 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 (continued)

Initials and Name	Definition
BHIZ K Branch on High Order Quantity 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. Turns on the overflow indicator 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. Turns on the overflow indicator in case of any overflow or underflow. b(VAC, . . . , VAC + 5) - c(K, . . . , K + 5) = c(VAC, . . . , VAC + 5)
BVSU K Backwards Vector Subtract △1 △5	Subtracts the vector contained in the VAC from the vector stored at K. Keeps the result in the VAC. Turns on the overflow indicator 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 (continued)

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 push-down list when no address is given.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

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. Turns on the overflow indicator 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. Turns on the overflow indicator 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 if the last one of an instruction string. This storage is at the last address of the address string or in the push-down list 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. Turns on the overflow indicator in case of overflow or underflow. [b(VAC, . . . , VAC + 5) DOT c(K, . . . , K + 5)] b(VAC, . . . , VAC + 5) = c(VAC, . . . , VAC + 5)

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

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. Turns on the overflow indicator 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. Turn on the overflow indicator 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. Turns on the overflow indicator 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). 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 (continued)

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). N is a number written into the program instead of an address. Keeps the shifted vector quantities in the VAC. Turns on overflow indicator 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. 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 pushdown list 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 (continued)

Initials and Name	Definition
BOV K Branch on Overflow △1 △6	Turns off the overflow indicator 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 DP before (prior) 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 DP before 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 DP before 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 (continued)

Initials and Name	Definition
ACOS DP Arccos Function (2)	Computes $1/2\pi$ arccos y where y is the DP before content of the MPAC. Keeps the DP result in the MPAC. The principal values lie between 0 and 1/2 (0 and π scaled). $1/2\pi$ arccos [2b(MPAC, MPAC + 1)] = c(MPAC, MPAC + 1)
DSQ DP Square (2)	Squares the DP before 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 before 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 before 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)

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

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. Turns on overflow indicator 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. Turns on the overflow indicator 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 (continued)

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 push-down list 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. $c(K, \ldots, K + 5) = c(VAC, \ldots, VAC + 5)$

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

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 push-down list, and V_3 from just below V_2 in the push-down list. 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.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

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 interpretive address N into the specified index register ($-37777 \leq N \leq +37776$). 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 interpretive address N into the specified index register ($-37777 \leq N \leq +37776$). 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 ($-37777 \leq N \leq +37776$). N is written into an address string instead of a relevant address K.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

Initials and Name	Definition
LXA, T K Load Index Direct from Address 7	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 7	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 7	Transfers the content of the specified index register to location K of E memory.
XCHX, T K Index Regis-ter Exchange 7	Exchanges the content of the specified index register with the content of location K of E memory.
XAD, T K Add to Index Register 7	Adds the content of location K in E memory to content of the specified index register. Turns on the overflow indicator in case of overflow or underflow.

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

Initials and Name	Definition
XSU, T K Subtract from Index Register ⚠	Subtracts the content of location K in E memory from the content of the specified index register. Turns on the overflow indicator in case of overflow or underflow.
AST, T N Address to Step True ⚠	Enters the quantity N into the specified step register ($-37777 \leq N \leq +37776$). N is written into an address string instead of a relevant address K.
TIX, T K Transfer on Index ⚠	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 ⚠	This instruction must be the last or only instruction of an instruction string. Causes the Interpreter to complete its turn immediately and to switch the AGC back to executing Basic Instructions in the basic mode.
RTB K Return to Basic at K ⚠	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 to 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 (continued)

Initials and Name	Definition
NOLOD No Load	Turns off the load indicator. This instruction can be used when the MPAC or VAC contains the desired quantity after execution of the last instruction of the previous instruction string.
LODON Load Indicator On	Turns on the load indicator. 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 (continued)

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.
ON N Turn Switch On or OFF N Turn Switch Off ⚠7	Sets switch N to ONE if the address word contains 10000 + N ($1 \leq N \leq 45$). Sets switch N to ZERO if the address word contains 30000 + N ($1 \leq N \leq 45$).
BSON N, K Branch on Switch ON or BSOFF N, K Branch on Switch OFF ⚠7	Executes next instruction at K if switch N is set to ONE or executes consecutive instruction if switch N is set to ZERO if address word contains 10000 + N ($1 \leq N \leq 45$). 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 30000 + N ($1 \leq N \leq 45$).

TABLE 6-1

INTERPRETIVE INSTRUCTION DEFINITIONS (continued)

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 push-down list 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 push-down list 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 (continued)

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 push-down list 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 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 push-down list 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	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	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	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
TSLT K	1	0	0	0	0	x	0
TSRT K	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	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	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	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.

TABLE 6-2

ORDER CODES (continued)

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 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.

TABLE 6-3

E REGISTER ASSIGNMENTS

Octal Address	Initials [1]	Name and Purpose
00100	BANKSET	Holds complemented number of the Bank in which the current program is stored.
101	ADDR (ADDRWD)	Holds real address (nine bits) in a Work Area. Holds any 10-bit address referring to E memory. Holds any 12-bit address for addressing F memory by means of register S. (Holds first Address Word in its true form temporarily.) Holds number N after decoding an Address Word.
102	ORDER	Holds Instruction Word to be decoded (in its true form). Holds second order code of a decoded Instruction Word. Holds 00000 if no second order code exists or processing of it has been started.
103	TEM11	Temporary storage used by subroutines DOT2 and INCRT4.
104	MODE	Holds 77776 when Interpreter is in DP mode. Holds 77775 when Interpreter is in TP mode. Holds 77777 when Interpreter is in Vector mode.
105	WORKLOC (FIXLOC)	Holds initial address of Work Area currently used. One out of five addresses is entered by the Executive Program.
106	VACLOC	Holds initial address of VAC currently used. VACLOC = WORKLOC + 32D. One out of five addresses is entered by the Executive Program.

[1] Initials in parentheses are used in MIT listings.

TABLE 6-3

E REGISTER ASSIGNMENTS (continued)

Octal Address	Initials [1]	Name and Purpose
00107-114	VBUF	Temporary storage (six locations) for vector operations.
107	TEMQS	Temporary storage used by subroutine SWCALL.
110	BANKTEM	Temporary storage used by subroutine SWCALL.
111	TEMB (B)	Argument storage used by subroutine ARCCOS.
113	TAG1	Holds initial address of Work Area currently used: plus zero or one dependent upon whether Index Register X1 or X2 is to be used, or plus two or three dependent upon whether Step Register S1 or S2 is to be used.
113	ESCAPE2	Negative Argument Switch. Contains a TC K instruction during execution of ARCCOS.
113	PUSHIND	Holds 00001 during VXSC K; 00000 during the execution of all other Interpretive Instructions.
114	AWORD (POLISH)	Holds an Address Word in its true form. Holds 10-bit E address of Store Code Address Word.
114	TEMQ3	Holds return address entered by subroutine DDV K and SQRTDIV.
115-117	BUF	Temporary storage (three locations) for DP and TP operations.
117	SGNDMAX	Temporary storage used by subroutine TPAGREE.

[1] Initials in parentheses are used in MIT listings.

TABLE 6-3

E REGISTER ASSIGNMENTS (continued)

Octal Address	Initials [1]	Name and Purpose
00120	TEM2	Temporary storage 2 used by subroutines TRAD, DAD1, STD2, DMP1, DDV, DOT2, INCRT2, STB, and CROSS1.
121	TEM4	Temporary storage 4 used by subroutines DMP1, INCRT4, and CROSS1.
121	TEM Q	Holds return address during execution of subroutine TPAGREE.
122	TEM5	Temporary storage 5 used by subroutines TRAD, DMP, DAD, and VACCOM.
122	TEM Q2	Holds return address during the execution of subroutine SQRT3.
122	BASE	Temporary storage used by subroutine CROSS1.
123	TEM6	Temporary storage 6 used by subroutine CROSS1.
123	TEM8	Temporary storage 8 used by subroutines STO2, DOT2, and ROUND.
124	TEM9	Temporary storage 9 used by subroutines POLY, MXV, and VXM.
125	TEM10	Temporary storage 10 used by subroutines POLY, MXV, and VXM.
125	IND	Temporary storage used by subroutine CROSS1.

[1] Initials in parentheses are used in MIT listings.

TABLE 6-3

E REGISTER ASSIGNMENTS (continued)

Octal Address	Initials [1]	Name and Purpose
00126	LOADIND (NEWEQIND)	Holds 00001 if MPAC or VAC has to be reloaded; holds 00000 if the present content of MPAC or VAC is to be operated with.
127-136	Job Area 1	As defined below.
127-131	MPAC	Multiprecision Accumulator (MPAC) (three locations).
132	IWLOC (LOC)	Holds address of Instruction Word to be decoded.
133	AWLOC (ADRLOC)	Holds address of last TC INTPRET. Holds address of last Instruction Word in current string. Holds address of any Address Word in current string.
134	OVFIND	Overflow Indicator. Holds normally 00000. Holds 00001 or 77776 in case of overflow or underflow.
135	PUSHLOC	Holds address of next available location in the pushdown list.
136	PRIORITY	Holds 77777 when this Job Area is available. When this Job Area is in use, the register holds a ZERO in bit position 15, the priority code of the Job using this Job Area in bit positions 14 through 10, and the address of the assigned Work Area in bit positions 9 through 1. Information is entered by the Executive Program.
137-146	Job Area 2	Similar to Job Area 1.

[1] Initials in parentheses are used in MIT listings.

TABLE 6-3

E REGISTER ASSIGNMENTS (continued)

Octal Address	Initials [1]	Name and Purpose
00147-156	Job Area 3	Similar to Job Area 1.
157-166	Job Area 4	Similar to Job Area 1.
167-176	Job Area 5	Similar to Job Area 1.
177-206	Job Area 6	Similar to Job Area 1.
207-216	Job Area 7	Similar to Job Area 1.
217-226	Job Area 8	Similar to Job Area 1.
227-233	LST1	Time List (five locations), holds times when next Tasks have to be executed.
234-241	LST2	Address List (five locations), holds addresses of Task programs to be executed.
242	EXECTEM1	Temporary storage used by EXECUTIVE.
243	EXECTEM2	Temporary storage used by EXECUTIVE.
244	EXECTEM3	Temporary storage used by EXECUTIVE.
245	VAC1USE	Holds 00000 when Work Area 1 is in use. Holds 00245 when Work Area 1 is available.
246-320	Work Area 1	As defined below.
246-305	PUSHLIST	Modified pushdown list (thirty-two locations).

[1] Initials in parentheses are used in MIT listings.

TABLE 6-3

E REGISTER ASSIGNMENTS (continued)

Octal Address	Initials [1]	Name and Purpose
00306-313	VAC	Vector Accumulator (VAC) (six locations) (Work Area address 0040).
314-315	X1, X2	Index Registers 1 and 2 (Work Area addresses 0046 and 0047).
316-317	S1, S2	Step Registers 1 and 2 (Work Area addresses 0050 and 0051).
320	QPRET	Holds return address.
321	VAC2USE	Holds 00000 when Work Area 2 is in use. Holds 00321 when Work Area 2 is available.
322-374	Work Area 2	Similar to Work Area 1.
375	VAC3USE	Holds 00000 if Work Area 3 is in use. Holds 00375 if Work Area 3 is available.
376-450	Work Area 3	Similar to Work Area 1.
451	VAC4USE	Holds 00000 if Work Area 4 is in use. Holds 00451 if Work Area 4 is available.
452-524	Work Area 4	Similar to Work Area 1.
525	VAC5USE	Holds 00000 if Work Area 5 is in use. Holds 00525 if Work Area 5 is available.
526-600	Work Area 5	Similar to Work Area 1.

[1] Initials in parentheses are used in MIT listings.

TABLE 6-3

E REGISTER ASSIGNMENTS (continued)

Octal Address	Initials [1]	Name and Purpose
00601	NEWJOB	A multiple of eight is entered (into bit positions 6 through 4) by Executive Program to signal Interpretive Rupt.
602-604	STATE	Holds 45-bit switch word entered by Interpreter (three registers).
605	BANKRUPT	Holds Bank Code for interrupted program entered by interrupting program.
606	NEWPRIO	Holds priority of new Job entered by EXECUTIVE.
607	WTEXIT	Temporary storage for Interrupt Programs.
610	LOCCTR	Temporary storage for Interrupt Programs.
611	DVSW	Divide Switch, holds 0000 for normal divide or any other quantity for backward divide.
612	BRANCH Q	Holds the return address for a branch operation.
613	COMPON	Component counter for UNIT operation.
614	ARETURN	Holds return address during the execution of Instructions ASIN and ACOS.
615	ESCAPE	Return Address Switch contains a TC instruction during the execution of Instructions ASIN and ACOS.
616	RUPTAGN	Temporary storage used by Waitlister.

[1] Initials in parentheses are used in MIT listings.