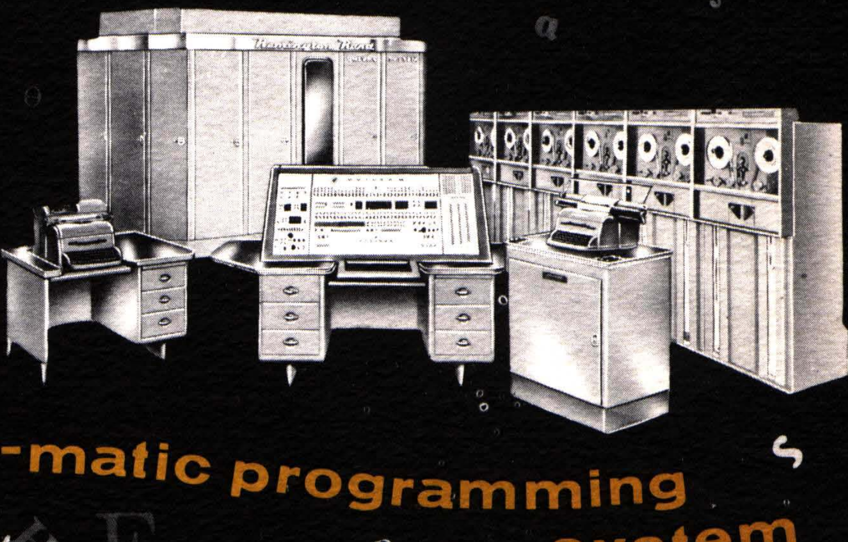
Remington Rand Univac
DIVISION OF SPERRY RAND CORPORATION



UNIVAC® math-matic programming system

UNIVAC® math-matic programming system

ANOTHER SERVICE OF . . .

MANAGEMENT SERVICES AND OPERATIONS RESEARCH DEPARTMENT

Remington Rand Univac

DIVISION OF SPERRY RAND CORPORATION

FOREWORD

THE UNIVAC II DATA AUTOMATION SYSTEM

The Univac II Data Automation System is a complete and well balanced data processing system. It will accept and prepare information through a wide variety of standard data-recording media. The user gains versatility most economically since the Central Computer, that unit which performs the actual processing, can read and write information directly through the magnetic tape which is one of the most rapid input-output media in use today. Peripheral equipments convert all recorded data into the form acceptable to the Central Computer, or from the form prepared by the Central Computer (Univac II System code on magnetic tape) to the desired form. In this way, the system has a dual advantage. First, the Central Computer need not be hampered in its processing task by the necessity of working directly with input-output media unworthy of its lightning-fast internal operating speeds. Secondly, the Central Computer need not be involved in conversion process which can most economically be handled by peripheral equipments on an off-line basis.

From a wide variety of available equipments each Univac II System user chooses the units which, when molded into a system, best meet his overall data processing requirements.

AVAILABLE EQUIPMENTS AND THEIR FUNCTIONS

The Univac II Central Computer

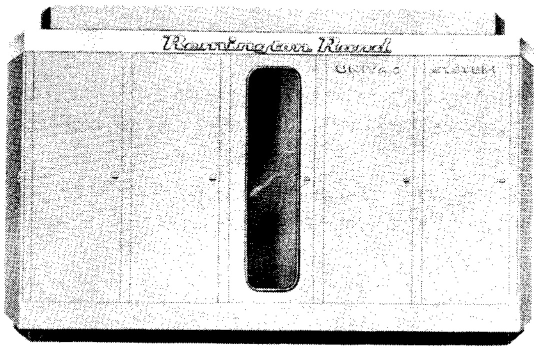


FIGURE 1

The Univac II Central Computer in Figure 1 is the heart of the Univac II Data Automation System. It performs all arithmetic and logical operations.

In the execution of a typical data processing task the Central Computer performs the following basic operations:

1. Step-by-step instructions, stating specifically the operations to be performed on the data, are

read by the Central Computer from magnetic tape and stored internally within the Central Computer. Obeying the stored instructions, the Central Computer then automatically ....

2. Reads the data from magnetic tape and stores it internally.

3. Performs all operations upon the data indicated by the instructions, and stores the results internally.

4. Reads the results from storage and writes them on magnetic tape.

All operations are self-checked to ensure that they are performed with the unwavering accuracy and dependability that has become associated with the name UNIVAC.

The Univac II Control Group

Two control units are directly connected to the Central Computer, and each in its own way, provides some indication of the actions of the Central Computer.

Univac Supervisory Control Console

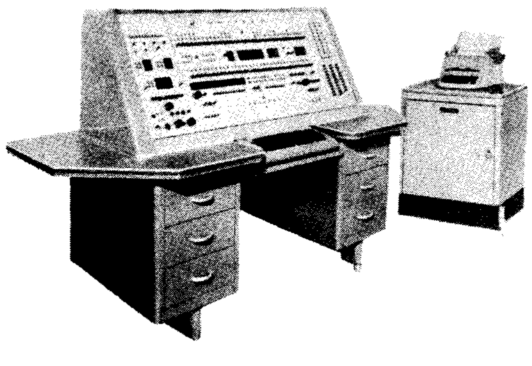


FIGURE 2

The Univac Supervisory Control Console (Figure 2) provides the operator with a continuous picture of the operations taking place within the Central Computer. It also provides visual indication whenever an error occurs in any operation, identifying the faulty circuit for the maintenance technician.

Although the Central Computer is designed to operate automatically, there are occasions when manual intervention may be desirable. The Univac Supervisory Control Console includes a keyboard by means of which the operator can type information directly into the Central Computer. A group of switches and buttons on the Console allows the interruption of automatic operations and the institution of changes in their course or the substitution or insertion of other operations.

Univac Supervisory Control Printer

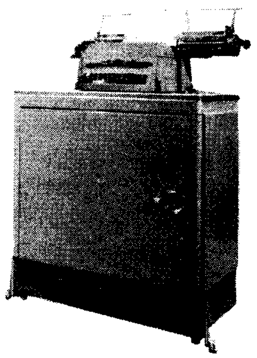


FIGURE 3

The Univac Supervisory Control Printer (Figure 3) is a modified electric typewriter which prints information directly from the Central Computer. Its primary function is to provide the operator, in easily readable form, information concerning the processing being performed within the Central Computer. This unit is sometimes employed for printing processing results; however, it is used for this purpose only when the information to be printed is not lengthy.

Univac Input Devices

The function of Univac II input devices is to convert information from its original form into Univac II System code recorded on magnetic tape. These devices are completely independent of the Central Computer, so that while the input devices prepare data for future use by the computer, the computer, itself, is free to carry on the current processing problems. This ability to overlap input preparation and computer processing represents a large saving of time and thus money, for the user.

Univac Unityper II

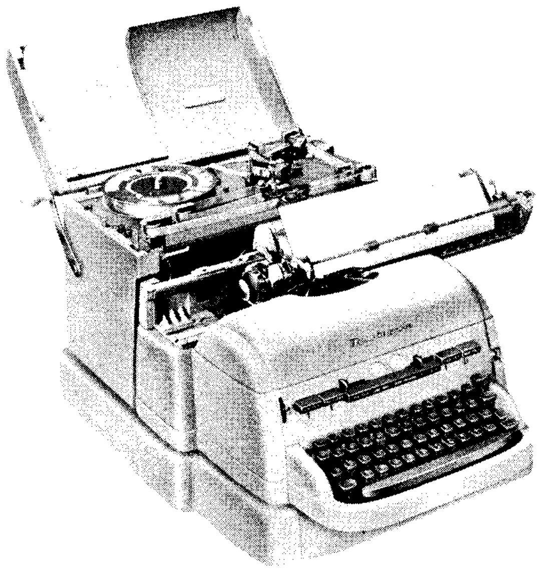


FIGURE 4

The Univac Unityper II (Figure 4) is a device by means of which information legible to its human operator can be recorded on magnetic tape. This device is somewhat larger than, though similar in appearance to, an electric typewriter. The 26 letters of the alphabet, 10 numerals, and some special Univac II System Symbols are represented on the keyboard of this device in an array similar to the familiar typewriter keyboard pattern. Striking a Unityper II key causes:

1. A pattern of magnetic spots representing the Univac II System Code for the character represented on the key to be recorded on a magnetic tape mounted in the upper portion of the device, and

2. That character to be printed on a piece of copy paper mounted on the carriage.

Thus, recording information on magnetic tape with the Unityper II involves little more than a retyping of the information. Information is tape-recorded by the Unityper II at a density of 50 characters per inch with a 2.4 inch spacing between each consecutive 120 characters.

Univac Verifier

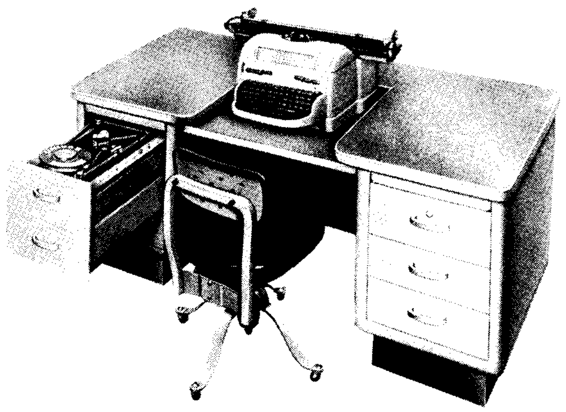


FIGURE 5

The Univac Verifier (Figure 5) is a unit of peripheral equipment which can operate in any one of two capacities:

1. As a primary input device which records information on magnetic tape by means of a typewriter keyboard in very much the same manner as the Unityper II.

2. As a proof reading device which corroborates information recorded on tape and permits the correction of detected errors.

Its primary use is as a proof reading and correcting device. Information is recorded by the Verifier at a recording density of 50 characters per inch with a 2.4 inch spacing between each 120 characters.

Univac Punched Card-to-Magnetic Tape Converter

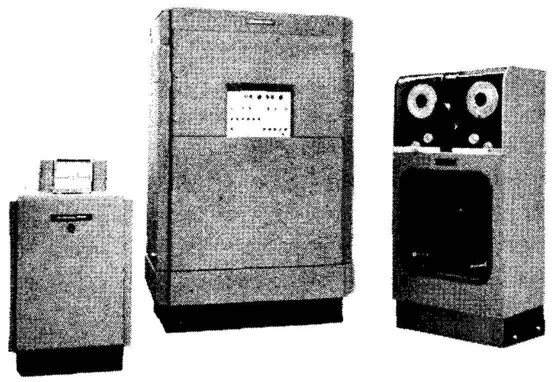


FIGURE 6

The Univac Punched Card-to-Magnetic Tape Converter (Figure 6) consisting of a card Reading Unit, a Control Unit and a Tape Unit, allows the entry of information into the Univac System in punched card form. Cards are loaded into the intake bin of the Card Reading Unit, and the information read from the cards is recorded on magnetic tape. The entire process is accomplished automatically and its operation is completely self-checked to ensure complete accuracy of the recorded information. The Converter is equipped with a removable plugboard which allows automatic rearranging of information during the conversion process.

Univac Punched Card-to-Magnetic Tape Converters are offered in two models. One handles standard 90-column punched cards; the other handles standard 80-column punched cards. Both models operate at a maximum conversion rate of 240 cards per minute and record information at a density of 128 characters per inch with a 1.8 inch space between each 120 characters, and a 2.4 inch space between each 720 characters.

Univac Paper Tape-to-Magnetic Tape Converter

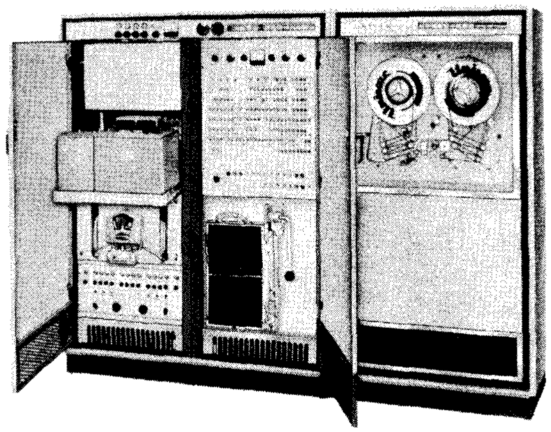


FIGURE 7

The Univac Paper Tape-to-Magnetic Tape Converter (Figure 7) is a device consisting of a Perforated Tape Reader, a Translator and Control Unit, and a Magnetic Tape Recorder. This equipment allows information recorded on paper tape to be entered directly into the Univac II System. Reels or message lengths of punched paper tape generated by teletypewriters, automatic typewriters, adding or bookkeeping machines with tape punchers attached, and punched card to perforated tape converters may be mounted on the Tape Reader. Information contained on tapes is automatically translated into Univac II System Code and recorded on magnetic tape. Deletion of certain punched paper tape symbols, and addition of some Univac II System Symbols may be accomplished automatically during the conversion process. The entire operation is completely self-checked to ensure complete accuracy of the conversion process. The Univac Paper Tape-to-Magnetic Tape Converter operates at a maximum conversion rate of 200 characters per second and records information at a density of 128 characters per inch, placing a one inch space between each 120 characters, and a 2.4 inch space between each 720 characters.

Univac Input-Output Devices

Univac Uniservo

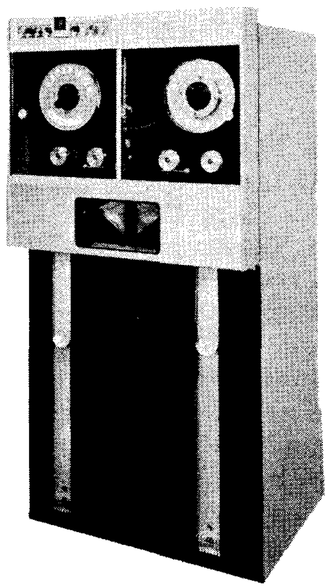


FIGURE 8

The Univac Uniservo (Figure 8) is the device through which the Central Computer communicates with its magnetic tapes. A maximum of 16 Uniservos may be directly connected to the Univac II Central Computer. Each Uniservo contains a "read-write" head and mechanism for moving the magnetic tape past the head at a speed of 100 inches per second. Each Uniservo is capable of reading tape moving in the forward direction, reading tape moving in the backward direction, writing on tape moving in the forward direction, and rewinding its tape. Reading from any one Uniservo, writing on any other Uniservo, rewinding the tape on any number of the remaining Uniservos may be carried on simultaneously with Central Computer processing. Uniservo operations are controlled by the Central Computer through programmed instructions.

Univac Output Devices

Univac II output devices allow the system to prepare processed results in a wide variety of forms. They automatically convert information contained on tapes produced by Central Computer processing into the desired form. All of these output devices operate with complete independence of the Central Computer. Thus, the computer is free to handle further processing while the results of the previous problem are being converted. This ability to overlap conversion and processing operations represents a great saving in time and money for the user.

Univac Uniprinter

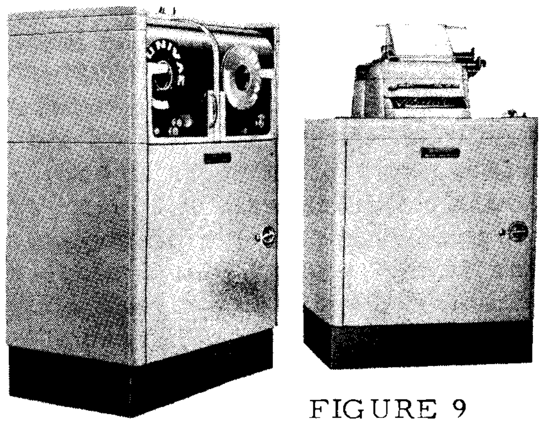


FIGURE 9

The Univac Uniprinter (Figure 9) consists of a Tape Reader and a Printing Unit which is a modified electric typewriter. A reel of magnetic tape, containing the information to be printed, is mounted on the Tape Reader. As information is read from the tape, it is printed by the electric typewriter. The Univac Uniprinter, which accepts tapes recorded at 25 characters per inch, prints at a rate of 10 characters per second, and is usually used for low volume output printing, such as the preparation of management reports.

Univac High-Speed Printer

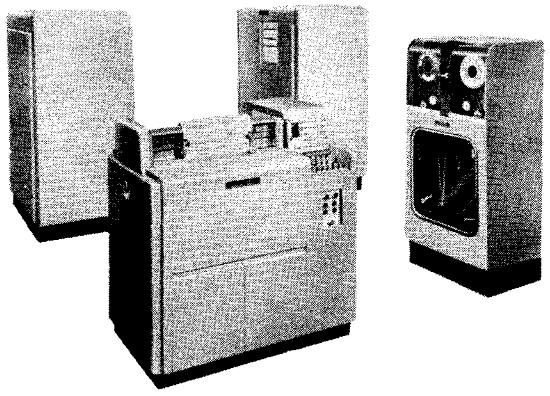


FIGURE 10

The Univac High-Speed Printer (Figure 10) is used for large volume printing. This four unit assembly, consisting of a Tape Reader, a Storage Unit, a control Unit, and a Printer, reads magnetic tape and converts the information recorded thereon into printed copy. The High-Speed Printer prints an entire line at a time. Each line may contain as many as 130 characters, and printing is accomplished at a maximum rate of 600 lines per minute. A removable plugboard mounted in the control Unit controls the format of the printed page and affords wide flexibility in the arrangement of the printed information, reducing the editing and thus the processing time required of the Central Computer. The entire operation of this device is completely self-checked to ensure that each character printed is the exact one recorded on the magnetic tape. It accepts information tape-recorded at a density of from 50 to 128 characters per inch with at least one inch space between each 120 characters.

Univac Magnetic Tape-to-Card Converter

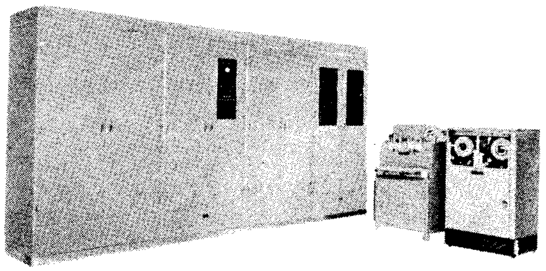


FIGURE 11

The Univac Magnetic Tape-to-Card Converter (Figure 11) consists of three units: a Tape Unit, a Card Punch Unit and an Electronic Cabinet containing the circuitry necessary to control and check the Tape and Card Punch Units. This piece of equipment reads information from magnetic tape and converts the information into standard punched cards at a rate of 120 cards per minute. A removable plugboard permits the selection and rearrangement of information during the conversion process. The Univac Magnetic Tape-to-Card Converter accepts information tape-recorded at a density of 128 characters per inch with at least one tenth inch space between each 120 characters and 2.4 inch space between each 720 characters. Its entire operation is completely self-checked to ensure proper conversion.

Univac Magnetic Tape-to-Paper Tape Converter

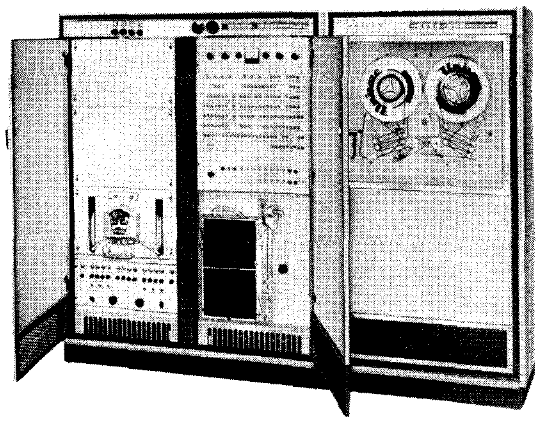


FIGURE 12

The Univac Magnetic Tape-to-Paper Tape Converter (Figure 12) consists of a Magnetic Tape Unit, a Translator and Control Unit, and a Paper Tape Punch. It punches information recorded on magnetic tape into paper tape. The punched paper tapes may then be used directly to send information via a teletypewriter.

As with all Univac II equipment the operation of the Magnetic Tape-to-Paper Tape Converter is completely self-checked to ensure accurate conversion. This conversion is accomplished at a maximum rate of 60 characters per second. It accepts information recorded at a density of 128 characters per inch with at least a 1 inch space between each 120 characters.

INTRODUCTION

The MATH-MATIC system was developed in response to the need for an algebraic language compiler for Univac. Today vast and complex mathematical problems may be solved in hours or minutes by the computer, once the many lines of computer code are prepared. The length of time consumed in learning to write a specialized computer code and then in producing and debugging a finished program was a major obstacle to efficiency. MATH-MATIC is designed to overcome this obstacle and make UNIVAC easily available to programmer and non-programmer alike for a wide range of mathematical applications. The system accepts as its input an orderly presentation of the problem in a pseudo-code which closely resembles English sentences and mathematical equations. Anyone who can state his problem logically can write a MATH-MATIC program for it. The system relieves the user of the burdensome details of storage allocation, keeping track of addresses, and arranging for segments of code on the running tape. The system has access to an extensive library of computer-coded subroutines for quickly calculating mathematical functions, and performing the necessary input-output and control operations. The system assembles these subroutines into a running program which will produce the desired results without further human intervention.

This manual is a complete practical guide for the user of the system. For details on the internal operation of the system and pertinent information on how to add to the subroutine library, see the MATH-MATIC PROGRAMMER'S MANUAL. (To be issued later). In the first chapter of the present manual, we have tried to impart over-all familiarity with the use of the system and its pseudo code. Chapter II follows with the specific formats and rules for the sentences and Chapter III contains the complete repertoire of MATH-MATIC expressions, and their formats and rules. This repertoire is extensive, and may be expanded very easily. Chapter IV provides all the information needed in the preparation of input data. Chapter V gives procedures to follow if a special problem goes beyond the present repertoire; in most cases, this chapter may be omitted. Chapter VI contains the operating instructions for a MATH-MATIC compilation and problem run, and other general information about the system. Normal and error printouts which occur during compilations and problem runs are found in the Appendix.

We feel that this approach will lead to quick and efficient utilization of the system by people with mathematical training who have had little or no programming experience. Those with more training in the programming field will find the flexibility and scope of the pseudo-code adequate to handle the most intricate problem.

I

WHAT IS MATH-MATIC PSEUDO CODE

MATH-MATIC pseudo-code is a set of words, numbers, and symbols arranged to give a complete, logical statement of a problem. The primary unit of pseudo-code is the sentence. There are three types of sentences; input-output, control, and equation. Input-output sentences cause data to be read into the memory from tape, or information in the memory to be written on tape. Control sentences determine the various paths taken through the program during running. Ordinarily, the sentences, which are numbered in ascending order, are executed in that order. Control sentences can alter this chain of execution in any way the user desires.

Equation sentences are stated as explicit algebraic equations subject to certain conventions listed in Chapter II. The left member of the equation must be the dependent variable, followed by an equal sign. On the right side of the equation appear the various mathematical functions of the independent variables that are needed to calculate the value of the dependent variable. Some examples of equation sentences follow;

```
(10) X = (15*Y+3*Z)/SIN A .

(6) X(I) = A(I)+B(J, I) .

(11) VARIANCE = SUMSQUARES/
     10-MEAN2 .
```

Input-output and control sentences take the form of English imperative statements. The first word of the input-output or control sentence tells the system the general type of command being given. The remaining words in the input-output or control sentence give further details about this command, and supply the names of relevant parameters and variables. Some examples of input-output and control sentences follow;

```
(1)  READ A B C .

(20) IF X > Y JUMP TO SENTENCE
     8 .

(12) EXECUTE SENTENCE 4
     THRU 8 .

(6)  VARY J 1 (1) 20 SENTENCE
     11 THRU 15 .
```

Certain conventions and rules regarding the insertion of spaces, parentheses, and periods into input-output and control sentences are necessary for the system to interpret the pseudo-code correctly. These rules are listed in detail in Chapter II. The user should note the flexibility in naming a variable; any single letter or combination of letters and numbers (starting with a letter) up to 12 digits can be handled. Constants may be stated as integers, fractions, decimals, or in power of ten form. One, two or three dimensional arrays of numbers can be read in from tape or constructed in memory. The elements of the array are referred to by the familiar notation of subscripts; these, in turn, may be *variables or constants. Four examples* follow in which typical problems are described and the Math-Matic pseudo-code statements of the problems are given.

Sample Problem 1:

Solve:

$$Y = \frac{X^3(2+X)}{3 \text{ Cos } A} - \sqrt[4]{3P}$$

for P running from 0.2 to 0.8 in increments of 0.2, A running from 0.35 to 1.05 in increments of .175 and X running from 1.8 to 3.8 in increments of 0.5

A MATH-MATIC pseudo-code statement of this problem is as follows;

```
(1) VARY P 0.2 (0.2) 0.8
    SENTENCES 2 THRU 5 .

(2) VARY A 0.35 (0.175) 1.05
    SENTENCES 3 THRU 5 .

(3) VARY X 1.8 (0.5) 3.8
    SENTENCES 4 THRU 5 .

(4) Y = X3*(2+X)/(3*COS A)-4
    ROOT (3*P) .

(5) WRITE AND EDIT Y X A P .

(6) STOP .
```

Sentence 1 will set P to its initial value, 0.2, and will insert following sentence 5 a control operation which will add the increment, 0.2, to P and return control to sentence 2. When P exceeds its limit value of 0.8, control will jump to the next operation following this control, in this case sentence 6. Sentences 2 and 3 will perform similar functions for A and X. The range components in these three sentences indicate that sentence 3 lies within the range of sentence 2, and sentence 2 lies within the range of sentence 1. This nesting of loops means that X, the variable of the innermost loop, will take on all of its values, for each value of A and P. When the value of X exceeds its limit value, A will be incremented and X will be reset to its initial value. P will be incremented and A and X will be reset, each time A exceeds its limit. In this way 100 values of Y will be computed and written. Sentence 6 will supply the necessary sentinels for the output and stop the program.

Sample Problem 2:

We have m samples of 10 values each of a statistical variate. A sentinel of Z's follows the last sample on the tape. We wish to calculate the mean and variance of each sample and edit the output for a uniprinter.

$$\text{MEAN} = \frac{\sum_{i=1}^{10} X_i}{10}$$

$$\text{VARIANCE} = \frac{\sum_{i=1}^{10} X_i^2}{10} - (\text{MEAN})^2$$

We will let the system allocate the input and output servos. The Math-Matic pseudo-code for this problem would be:

```
(1) READ-ITEM X(10) IF SENTINEL
    JUMP TO SENTENCE 11 .

(2) SUM = 0 .

(3) SUMSQUARES = 0 .

(4) VARY I 1 (1) 10 SENTENCE 5
    THRU 6 .

(5) SUM = SUM+X(I) .

(6) SUMSQUARES = SUMSQUARES+
    X(I)2 .

(7) MEAN = SUM/10 .
```

```
(8) VARIANCE = SUMSQUARES/10-
    MEAN²

(9) WRITE AND EDIT FOR UNIPRIN-
    TER MEAN VARIANCE .

(10) JUMP TO SENTENCE 1 .

(11) STOP .
```

Sentences 1 and 9 are input-output statements; sentences 4, 10, and 11 are control statements. The rest are all equations. The 10-quantity array read in by sentence 1 will occupy 20 words of storage, since all numbers inside the Math-Matic system are in 2 word floating decimal form. Therefore, every time sentence 1 is executed, the next 20 word item is moved to the current item position. These items are automatically read from the input tape in 60 word blocks, whenever necessary. The user should consider the input items available one at a time as the READ sentence is executed.

The VARY in sentence 4 gives I an initial value of 1 and inserts after sentence 6 a routine which will increment I by 1 and return control to sentence 5, until I exceeds the limit, 10; then control passes to sentence 7. That is, sentences 5 and 6 are executed 10 times, with I = 1, 2, . . . , 10. The system automatically places a sentinel on the output tape following the last valid output item, checks the readability of the output tape, and prints out the number of blocks of output on the tape. The superscript symbol 2 which appears in sentences 6 and 8 can be unityped. The method of unityping them is discussed later in the chapter.

Sample Problem 3:

A is a 50X12 matrix, and B is a 12X20 matrix. We want to calculate matrix D = AxB. We wish to produce two sets of output, one edited for printing and the other unedited for future calculations. We also wish to print out the element of D with the largest absolute value. We prepare the 240 elements of B on a tape arranged continuously, one row after another. The individual rows of A follow on the same tape and a sentinel of Z's is placed in the first word after the last valid item. We have arbitrarily chosen Servo 3 for the input tape, and we will allow the system to select the output servos for us. The following is a Math-matic pseudo-code for this problem;

```
(1) READ-ARRAY B(12, 20) SERVO
    3 .

(2) LARGEST = 0 .

(3) READ-ITEM A(12) SERVO 3 IF
    SENTINEL JUMP TO SENTENCE
    15.

(4) VARY I 1 (1) 20 SENTENCE 5
    THRU 9.

(5) D(I) = 0 .

(6) VARY J 1 (1) 12 SENTENCE 7 .

(7) D(I) = D(I)+A(J)*B(J, I) .

(8) IF |D(I)| > LARGEST, JUMP TO
    SENTENCE 13.

(9) IGNORE .

(10) WRITE-ARRAY D(20) .

(11) WRITE-ARRAY CONVERTED
     D(20)

(12) JUMP TO SENTENCE 3 .

(13) LARGEST = D(I) .

(14) JUMP TO SENTENCE 9 .
```

```
(15) PRINT-OUT LARGEST .

(16) STOP .
```

Sentences 1, 3, 10, and 11 are input-output sentences; 2, 5, 7, and 13 are equations, and the rest are control sentences. Sentence 1 reads in the entire B matrix; sentence 3 reads in one row at a time of the A matrix. When the sentinel item of A is reached the computer shifts to finishing up operations. Sentence 2 gives an initial value to the current largest element. Sentences 8, 13, and 14 keep "LARGEST" up-to-date by substituting for it any element whose absolute value is greater. Sentence 5 gives each "D" element a starting value of zero before the summation in sentence 7 calculates the true value. Sentence 4 gives I an initial value of 1, and places after sentence 9 a routine which will increment I by 1 and return to sentence 5 until I exceeds the limit, 20; then this routine lets control pass to the next sentence. That is, sentences 5 to 8 are repeated for I = 1, 2, ... 20. Similarly, sentence 7 is repeated for J = 1, before control moves on to sentence 8. Sentence 15 prints out, in floating decimal form, the largest element of D and sentence 16 tells the system the problem is finished. The system then automatically checks the readability of the information on the output tapes and prints out the number of blocks on each. The word "CONVERTED" in sentence 10 means edited with a properly placed decimal point. The system assumes WRITE's (WRITE-ITEM's WRITE-ARRAY's) are for high speed printer unless otherwise specified.

Sample Problem 4:

We have 2 input tapes; one contains n sets of values of A, B and C with a sentinel of Z's after the last set. Beginning with the block after the sentinel, this same tape contains m sets of values of F, G, H and N. Again a sentinel follows the last set. The other input tape contains p sets of values of D and E; p = max(m, n). We wish to evaluate X_1 where,

$$X_1 = \frac{(7000*Y*A*\sin\alpha)^3}{B^D + C^E}$$

where the value of α is typed in during the problem run. Y assumes all the values 1.0, 1.1, 1.2, ... 3.0 for each set of values of A, B, C, D, and E, and the first set of values of D and E goes with the first set of A, B, and C.

We also wish to evaluate X_2 where,

$$X_2 = \frac{\sqrt[3]{E-G+\log_{10}(D+N)}}{F^{2.6}*e^H}$$

where the first set of values of D and E now goes with the first set of values of F, G, H, and N. Edited output is desired, consisting of A, Y, D, E and X_1 in the first case, and F, D, E, and X_2 in the second. Only one output tape is to be used.

The following is a Math-Matic pseudo-code for this problem;

```
(1) TYPE-IN ALPHA .

(2) READ A, B, C SERVO 4 STORAGE
    A IF SENTINEL JUMP TO SEN-
    TENCE 8 .

(3) READ D E SERVO 5 .

(4) VARY Y 1 (0.1) 3 SENTENCE 5
    THRU 6 .

(5) X1 = (7*10^3*Y*A*SIN ALPHA)^3/
    (B POW D+C POW E) .

(6) WRITE AND EDIT  A, Y, D, E X1
    SERVO 6 .

(7) JUMP TO SENTENCE 2 .
```

```
(8) CLOSE-INPUT AND REWIND
    SENTENCE 3 .

(9) CLOSE-OUTPUT SENTENCE 6 .

(10) READ F G H N SERVO 4
     STORAGE A IF SENTINEL
     JUMP TO SENTENCE 15 .

(11) EXECUTE SENTENCE 3 .

(12) X2 = (3 ROOT (E-G)+LOG (D+N)/
     (F^2.6*EXP H) .

(13) WRITE EDIT F D E X2 SERVO 6 .

(14) JUMP TO SENTENCE 10 .

(15) STOP .
```

Sentences 1, 4, 7, 14 and 15 are control statements. Sentences 5 and 12 are equations; the rest are input-output statements. Sentence 8 will rewind servo 5 and reset Sentence 3 so that the next time it is executed the first D, E pair will be brought into storage. Sentence 9 clears whatever output items remain in memory on to servo 6 and places a sentinel on that tape, so that another output may be written on it. Sentence 11 causes sentence 3 to be executed, after which control goes to sentence 12. It would have been permissible, and more efficient to repeat the READ instruction of sentence 3 at sentence 11. READ's in this pseudo-code are ordinary READ's, not READ-ITEM's or READ-ARRAY's because the variables are listed singly rather than being grouped in subscripted arrays.

A number in power of ten form appears at the beginning of sentence 5 and a decimal exponent appears at the end of senfence 12. The unityping of these superscript exponents is discussed in the next section of this chapter. The specific rules for the expressions in control, input-output, and equation sentences appear in the next chapter.

The letters, digits and symbols appearing in the sentences of the sample problems are all on the ordinary unityper keyboard in the same form except for the superscripts and the greater than (>) and less than (<) symbols. These 15 symbols (including superscript minus, decimal point and slash) have equivalents on the ordinary keyboard whose pulse patterns will be correctly interpreted by the system. For example, superscript 2 is represented by a "£" on the keyboard. A list of these equivalents appears in the appendix to this manual. However, a modified keyboard may be obtained from Remington Rand Univac which will produce the intended superscript or other special symbol on the typed copy. The use of the list of equivalent for >, <, and numerical exponents brings every MATH-MATIC pseudo-code within range of the ordinary Unityper keyboard. No special training is required to type pseudo-code or to check it.

The experienced programmer will occasionally desire to write a program for a special problem which goes beyond the present MATH-MATIC repertoire. There are two types of code other than MATH-MATIC pseudo-code which may be employed; the intermediate code of MATH-MATIC, known as Arith-Matic pseudo-code, and the familiar Univac code, called C-10. A group of lines of Arith-Matic pseudo-code may be inserted into the body of a MATH-MATIC pseudo-code by means of a "COMPILER" sentence. A "COMPUTER" sentence causes a similar insertion of any number of lines of Univac C-10 code. These are discussed thoroughly in Chapter V. Both COMPUTER and COMPILER sections may refer to any variable appearing elsewhere in the problem or to any constant, by means of a directory. Examples and rules of the two sections and the directory appear in Chapter V. Because of the flexibility and scope of the MATH-MATIC pseudo-code, these special sections will rarely be needed, and the Chapter on them may be safely omitted by those not familiar with UNIVAC.

II

HOW TO WRITE MATH-MATIC PSEUDO-CODE

Certain formal rules must be followed in writing Math-Matic pseudo-code. These rules are the minimum consistent with the two objectives of the pseudo-code; avoiding logical ambiguity in the system and stating the problem intelligibly for the future reference of the user. Those rules and formats which apply to particular sentences and functional call words are given in Chapter III along with a complete repertoire of the sentences and functional call words.

All rules and formats must be strictly adhered to, or error printouts or misinterpretations by MATH-MATIC will result. Some of the errors can be rectified by type-ins but the user should not depend on this. Thorough study of the rules, and a careful check of the pseudo-code are strongly recommended for efficient use of compiling time. We repeat the last example of the previous chapter with some minor changes in the numbering of the sentences.

(2) TYPE-IN ALPHA .

(2A) READ A B C SERVO 4 STORAGE A IF SENTINEL JUMP TO SENTENCE 8.

(3) READ D F SERVO 5 .

(4) VARY Y 1 (0.1) 3 SENTENCE 5 THRU 6 .

(5) X1 = (7*10^3*Y*A*SIN ALPHA)^3/ (B POW D+C POW E) .

(6) WRITE AND EDIT A Y D E X1 SERVO 6 .

(7) JUMP TO SENTENCE 2A .

(8) CLOSE-INPUT AND REWIND SENTENCE 3 .

(9) CLOSE-OUTPUT SENTENCE 6 .

(10) READ F G H N SERVO 4 STORAGE A IF SENTINEL JUMP TO SENTENCE 20.

(11) EXECUTE SENTENCE 3 .

(12) X2 = (3 ROOT (E-G)+LOG (D+N))/ (F$^{2.6}$*EXP H) .

(13) WRITE EDIT F D F X2 SERVO 6 .

(16) JUMP TO SENTENCE 10 .

(20) STOP .

The following rules apply to all sentences:

1. Every sentence must start with a sentence number enclosed in parentheses and followed by a space. The sentence number may be any integer from 1 to 999, with or without an appended alphabetic. e.g., (65) , (270) , (3A) , (91Q) , (810N).

2. The sentence numbers must be in ascending order, but need <u>not</u> be consecutive. (Note the last two sentences of the example). Sen-

tence 3 precedes sentence 3A, and sentence 3A precedes sentence 4.

3. When writing MATH-MATIC pseudo-code, the user may ignore entirely the positions of the words and sentences within UNIVAC words and blockettes. Sentence, words and names of variables may begin in any part of a word or blockette, and may overlap the following word or blockette.

4. Every sentence must end with a space, followed by a period, "Δ. ". Any number of spaces may appear directly before the period and any number of spaces may come between the period of one sentence and the left parenthesis of the next sentence number. In the example, each line is filled with spaces after the period, so that the sentences may appear one under another.

5. The last sentence in a pseudo-code must be "ΔSTOPΔ. ".

6. When sentence numbers appear as references, as in sentence 5, 8, and 9, they should not be surrounded by parentheses.

7. A decimal quantity with no integral part should be stated with a zero preceding the decimal point; e. g., 0.3561, and 0.0024, not .356 and .0024. This rule avoids the unintentional use of a space-period combination which would tell the system to end the sentence.

8. A variable may be any letter, or any combination of letters and digits beginning with a letter. A variable may not consist of more than 12 digits and letters.

9. A constant may appear as an integer, decimal, fraction, or in power of ten notation. 300, 300.00, 3000/10 and $3*10^2$ are all legitimate forms of the same number. No more than 11 digits may be written in any one number.

<u>RULES FOR WRITING EQUATIONS</u>

1. Every equation must begin with ΔXΔ=Δ. . ., where "X" is any one variable (with or without subscripts). Spaces must surround both the variable and the equal sign.

2. The rest of the equation consists of operation signs, functional call words, variables, numbers, and exponents. Spaces must not be used except to separate functional call words from their arguments, as "SINΔALPHA" and "BΔPOWΔD" in sentence 5.

3. Allowable operation signs are +, -, *(Multiplication), /(division), |. . | (absolute value sign), and pairs of parentheses, which have the usual algebraic significance of causing the operations inside them to be performed first. One equation may contain up to 100 operation signs, functional call words and exponents.

4. The asterisk (*) for multiplication must appear wherever a product is desired. The system cannot infer a product from the position of the number and variables. e.g., 5*A must be written, not 5A or 5. A.

5. Whenever in the absence of parentheses a choice exists regarding the order in which to perform indicated operations (e.g., should

A*B+C be considered (A*B)+C or A*(B+C)?), the system gives priority to these 3 classes in order.

a. functional call words and exponentiation.

b. * and /.

c. + and -.

Within each class the order of execution is left to right. Two brief examples will illustrate:

X/Y*Z means (X/Y)*Z

LOGΔA*B+C means (B*LOGΔA) + C.

6. Parentheses should always be used to clarify the order of execution. No harm results from the inclusion of unnecessary parentheses as long as they are properly paired.

7. Positive or negative numerical exponents are expressed directly as superscripted integers, decimals or fractions; e.g., X^5, X^{-3}, $X^{3.47}$, $X^{3/5}$. Literal exponents require functional call words, like "POW" in sentence 5.

8. Any variable appearing on the right side of an equation must have been given a value by a previous sentence. Y, ALPHA, A, B, C, D and E are right hand variables in sentence 5. Y was defined by the VARY in sentence 4, ALPHA by the TYPE-IN of sentence 1, and A through E by the READ's of sentences 2A and 3. Sentence 5 itself defines X1.

9. The one variable on the left side, or any of the variables on the right side, may be subscripted as long as the array (from which an element is being selected) is defined elsewhere by a READ-ARRAY, READ-ITEM, WRITE-ARRAY, WRITE-ITEM or CONTAIN sentence.

10. The MATH-MATIC system will consider a variable subscripted if a left parentheses immediately follows the variable. The subscripts are assumed to follow in order, separated by commas, with a right parenthesis following the last subscript. The subscripts may be constants, variables, or functions of variables involving only the four basic arithmetic operations.

Examples:

X(I, J, K)	Y(3*A+1, 5)
X(J, 3, I)	Y(A+B+C+2, D)
X(2*I)	Y(Q*R)

11. The subscripted variable, including the parentheses and the subscript functions may not exceed 12 digits.

<u>Rules for input-output sentences</u>

1. If the variables are listed individually, as in sentences 2A, 3, 6, 10, and 13, a READ or a WRITE sentence must be used. If the variables appear as a subscripted array, as in the second sample problem of the previous chapter, a READ-ARRAY„ READ-ITEM, WRITE-ARRAY, WRITE-ITEM, or CONTAIN sentence must be used. These are called the subscript input-output sentences.

2. In any ordinary input-output sentence, up to 50 variables may be listed.

3. The subscripts in a subscript input-output sentence must be integers, and not variables. There may be one, two or three subscripts; their product, which determines the number of elements in the array, must not exceed 250.*

4. SERVOΔ(1 through 9 or -) or STORAGEΔ (A through Z, but not W) or both SERVOΔ and STORAGEΔ, either one first, may be specified in any input-output sentence. Both specifications must appear after the list of variables, or after the subscript array.

5. The system will assign a new servo to any input or output sentence in which no servo is specified. Since one servo must be reserved for the running program tape, a maximum of nine servos are available to the input-output sentences. Unless the user desires to give two input-output sentences the same servo, there is no need to specify servos.

6. The user may put several inputs on the servo, several outputs on one servo, or he may use an output as an input later in the problem. He may also use the same input more than once. To accomplish any of these duplications, the same servo must be specified in the sentence involved. Also the appropriate "IF SENTINEL" option or CLOSE-INPUT or CLOSE-OUTPUT sentence must be used to position tapes or insert sentinels. See the next chapter for details.

7. The system will assign an adequate storage area to every input-output or CONTAIN sentence regardless of whether "STORAGE" is specified in the sentence. There is a maximum of 660 words (330 quantities) available for storage However, the system requires an indeterminate amount of storage for constants and partial results. The user must establish a safe limit under 660 words for all input-output and CONTAIN sentences combined. An error print-out during compilation will inform the user if the pseudo-code has exceeded the limit of 660 words. If the Univac II Master tape is employed, the storage maximum is increased to 1700 words.

8. If two input, output, or CONTAIN sentences can be safely assigned the same storage area, then the user should specify the same storage because this will save memory space and increase tape handling efficiency. Sentence 2 and 10 of the sample problem at the beginning of this chapter illustrate this point.

9. Except in the case of edited output, the storage letter named by the user applies to the entire area that the system reserves for the sentence. Where 15 or fewer quantities are being edited, a storage letter named by the user applies to the edit block, and not to the storage area of the quantities. Where 16 or more quantities are being edited, the storage letter applies to the storage area of the quantities, and not to the edit block. In this case one edit block is shared by all edit sentences (of 16 or more quantities) regardless of storage designations by the user.

10. Whenever two output sentences of 16 or more quantities, edited or unedited, have the same list of variables or the same subscript array, the system assigns the same storage area to the set of quantities in the two sentences un-

*For UNIVAC II, this limit is increased to 800.

less the user specifies different areas.

11. ΔIF SENTINEL JUMP..., IF SENTINEL RESET AND JUMP..., or ΔIF SENTINEL REWIND AND JUMP..., option may be included in any READΔ or READ-ITEMΔ sentence. If present, this phrase must appear at the end of the sentence.

12. The dash in the middle of the expressions READ-ARRAY, READ-ITEM, WRITE-ARRAY, WRITE-ITEM, CLOSE-INPUT, and CLOSE-OUTPUT must not be omitted or changed to a space.

13. A space must appear between all words and all variables in any input-output sentence. Any number of unnecessary spaces may be included, and commas may be substituted for spaces at any time.

14. A maximum of 40 input-output sentences in one pseudo-code is permitted..

15. In any WRITE, WRITE-ITEM, or WRITE-ARRAY sentence, whether edited or not, the output will be written for the high speed printer unless the phrase "FOR UNIPRINTER" appears in the sentence.

16. "CONVERT" and "FOR UNIPRINTER" may not appear in the same output sentence.

Rules governing control sentences

1. All words, variables, and signs of comparison must be preceded and followed by spaces. Commas or semicolons may be substituted for spaces at any time, except immediately following the first word in the sentence.

2. A variable used as a subscript may appear in any sentence whose format includes variables. X(3, 4) and X(I, J) are subscripted variables. I and J in the latter expression are variables used as subscripts. Subscripted variables may appear only in IF and PRINT-OUT sentences.

III

MATH-MATIC REPERTOIRE

The first part of this chapter is devoted to a complete list of formats of the functional call words, control sentences, and input-output sentences presently available to the MATH-MATIC system. In this list no attempt has been made to indicate all the options and parameters allowed in each call word or sentence. In the second part of this chapter, containing the functions and rules of each call word and sentence, the range of allowable options and parameters is discussed in detail. A list in the appendix shows every individual option available in the pseudo-code. The user is advised to refer to Chapter IV on "Data Preparation and Formats" when writing input-output sentences in pseudo-code. Otherwise, the discussions of functions and rules of call-words and sentences and the general rules given in Chapter II contain all the necessary information for writing pseudo-code.

FUNCTIONAL CALL WORDS

TRIGONOMETRIC FUNCTIONS

Pseudo-code Call-word	Function
1. SIN∆A	sine A
2. COS∆A	cosine A
3. TAN∆ALPHA	tangent α
4. COT∆A	cotangent A
5. SEC∆B	secant B
6. CSC∆B	cosecant B
7. ARCTAN∆X	$\tan^{-1} X$

HYPERBOLIC FUNCTIONS

Pseudo-code Call-word	Function
8. SINH∆A	sinh A
9. COSH∆A	cosh A
10. TANH∆A	tanh A

EXPONENTIAL AND LOGARITHMIC FUNCTIONS

Pseudo-code Call-word	Function
11. A∆POW∆B	A^B
12. A∆INTPOW∆B	A^B (B integral)
13. N∆ROOT∆A	$\sqrt[N]{A}$
14. SQR∆A	$\sqrt{A}$
15. EXP∆A	e^A
16. LOG∆A	$\log_{10} A$
17. LN∆A	$\log_e A$

CONTROL SENTENCES

1. CONTAIN X(m, n).
2. EXECUTE SENTENCE F THRU L.
3. IF X = Y JUMP TO SENTENCE F.
4. JUMP TO SENTENCE F.

5. PRINT-OUT X Y Z.

6. TYPE-IN A B C.

7. VARY X X_0 (X_i) X_L SENTENCE F THRU L.(increment type).

8. VARY X Y Z $X_0, Y_0, Z_0, X_1, Y_1, Z_1, \ldots, X_n, Y_n, Z_n$, SENTENCE F THRU L.(list type).

9. IGNORE.

10. STOP.

INPUT-OUTPUT SENTENCES

1. READ A B C.
2. READ-ARRAY X(m, n).
3. READ-ITEM X(m, n, p).
4. WRITE A B C.
5. WRITE-ARRAY X(m, n, p).
6. WRITE-ITEM X(m, n).
7. WRITE EDIT A B C.
8. WRITE-ARRAY EDIT X(m, n).
9. WRITE-ITEM EDIT X(m, n, p).
10. CLOSE-INPUT SENTENCE F.
11. CLOSE-OUTPUT SENTENCE F.

FUNCTIONAL CALL WORDS

Trigonometric Functions:

Call word	Function
SIN∆A	sine A
COS∆A	cosine A
TAN∆A	tangent A
COT∆A	cotangent A
SEC∆A	secant A
CSC∆A	cosecant A
ARCTAN∆A	arctan A

Rules:

1. A is expected to be in radians, except in the case of ARCTAN∆A, where the result will be in radians.
2. If A is a function rather than a single variable, it must be in parentheses, e.g., SIN∆(A+B), TAN∆(2*A).
3. The user is responsible for seeing that the parameters of functional call words do not take on values during the problem run which would cause the function to become infinite, such as A=0 in CSC∆A. An error printout will occur and the computer will stop.

Hyperbolic Functions:

Call-word	Function
SINH∆A	sinh A
COSH∆A	cosh A
TANH∆A	tanh A

Rules:

1. A is expected to be in radians.
2. If A is a function rather than a single variable, it must be in parentheses e.g., SINH∆(A+B*C).

Exponential and Logarithmic Functions:

Call-word	Function
A∆POW∆B	A^B
A∆INTPOW∆B	A^B
N∆ROOT∆A	$\sqrt[N]{A}$
SQR∆A	$\sqrt{A}$
EXP∆A	e^A
LOG∆A	$\log_{10} A$
LN∆A	$\log_e A$

Rules:

1. B, with the call word INTPOW, must take on integral values, with $|B| < 100$. N must be a positive integer or must take on integral values, with $N < 10$.

2. If A, B, or N is a function rather than a single variable or constant, it must be in parentheses: e. g. , (3*A)∆POW∆(B+C), EXP∆($(A)^2$+$(B)^2$).

3. The user is responsible for seeing that the parameters of functional call words do not take on values during the problem run for which the function is infinite or underfined, such as A=-3 in SQR∆A or LOG∆A. An error printout will occur and the computer will stop.

CONTROL SENTENCES

CONTAIN

Formats:

(n) CONTAIN X(m, n).

(n) CONTAIN X(m, n, p) STORAGE A.

Example:

(5) CONTAIN X(20) STORAGE A.

Functions:

This sentence causes MATH-MATIC to set aside adequate storage for the subscript array in the sentence. If the elements of a subscript array are referred to in an equation without the subscript array appearing in any READ-ITEM, READ-ARRAY, WRITE-ITEM or WRITE-ARRAY sentence, a CONTAIN sentence must be written for the subscript array.

Conventions:

1. The CONTAIN sentence may appear anywhere in the pseudo-code before the STOP sentence.

Rules:

1. The subscript array in a CONTAIN sentence may imply from one up to 250 elements. A list of variables may not appear in a CONTAIN sentence.

EXECUTE

Formats:

(n) EXECUTE SENTENCE F.

(n) EXECUTE SENTENCES F THRU L.

Example:

(15) EXECUTE SENTENCE 23 THRU 30.

Function:

When sentence n is executed, control is transferred to sentence F. In the "EXECUTE SENTENCE F" case, after sentence F is executed, control is transferred to the sentence following n. In the "F THRU L" case, after sentence L is executed, control is transferred to the sentence following n.

Conventions:

1. The ranges of two EXECUTE sentences may overlap each other.

2. If two EXECUTE ranges have the same last sentence, the one executed, last during running determines what sentence control returns to after sentence L is executed. An example follows:

   (2) EXECUTE SENTENCE 5 THRU 10.

   (3) ....

   (4) ....

   (5) ....

   (7) IF A > B JUMP TO SENTENCE 12.

   ...

   (10) ....

   (12) EXECUTE SENTENCE 10.

   (13) ....

If at sentence 7 control is transferred to sentence 12, then after sentence 10 is executed, control returns to sentence 13, not sentence 3.

Rules:

1. Sentence n, F and L may occur in any order in the pseudo-code but in order of execution F must precede L and n must be outside the range F to L.

IF

Formats:

(n) IF X > Y JUMP TO SENTENCE F.

(n) IF X < Y JUMP TO SENTENCE F.

(n) IF X = Y JUMP TO SENTENCE F.

(n) IF X > = Y JUMP TO SENTENCE F IF V < W JUMP TO SENTENCE G.

(n) IF X = < Y JUMP TO SENTENCE F IF V = W JUMP TO SENTENCE G IF P > Q JUMP TO SENTENCE H.

Examples:

(6) IF Z > = 10 JUMP TO SENTENCE 13.

(9) IF A(7, 7) < B(7, 7) JUMP TO SENTENCE 11 IF X > = 7.175*10^{-9} JUMP TO SENTENCE 17.

(39B) IF Z > W JUMP TO SENTENCE 24C.

(107Q) IF P(2, 4, 4) = > 6.66479*10^{11} JUMP TO SENTENCE 79.

(41) IF A > B JUMP TO SENTENCE 47 IF H < 0.759431 JUMP TO SENTENCE 23A IF A = B JUMP TO SENTENCE 26.

Functions:

When an "IF" sentence with one "if" clause is executed, a test is performed for the indicated condition. If the condition is met (e.g., in the first example, if X is greater than Y) control is transferred to sentence F. If the condition is not met, control moves to the next sentence after n. If there is more than one clause in the "IF" sentence, the conditions indicated in the clauses are tested in the order in which the clauses are written. With each one, if the condition is met, control is transferred to sentence F (or G, H, etc.). If the condition is not met, the next clause is tested until the clauses are exhausted. Then control moves to the next sentence after n. The phrase "IFΔXΔ>Δ=ΔYΔ" in pseudo-code means "IF X $\geq$ Y" or if X is equal to or greater than Y.

Conventions:

1. Any number of "if" clauses may be included in an "IF" sentence.
2. The two quantities being compared in an if clause may be any ordinary or subscripted variable with or without absolute value signs or any integer, decimal, or power of ten form.
3. The comparison symbol between the two compared quantities may consist of any one or any two of the three symbols, <, =, or >. If two of these symbols are used, they may appear in either order.
4. If either of the compared quantities has absolute value signs, MATH-MATIC takes the absolute value of both when it performs the comparison; e.g., "IFΔ|X|Δ>ΔYΔ" functions like "IFΔ|X|Δ>Δ|Y|Δ." "IFΔ|X|Δ>20Δ" functions the way it appears.

Rules:

1. The sentences F, G, H, etc., may come before or after sentence n, but they must appear somewhere in the pseudo-code.

JUMP

Format:

(n) JUMP TO SENTENCE F.

Example:

(10) JUMP TO SENTENCE 3A.

Function:

When this sentence is executed, control is transferred to sentence F.

Rules:

1. Sentence F may come before or after sentence n but it must appear somewhere in the pseudo-code.

PRINT-OUT

Format:

(n) PRINT-OUT A B C...N.

Example:

(10) PRINT-OUT ALPHA BETA X(K).

Function:

When this sentence is executed, the values of listed variables will be printed out on the supervisory control printer in two word floating decimal form. See Chapter IV on "Formats" for a discussion of two word floating decimal form.

Rules:

1. Any number of ordinary or subscripted variables up to 50 may be listed in a PRINT-OUT sentence.

TYPE-IN

Format:

(n) TYPE-IN A B C...N.

Example:

(10) TYPE-IN X Y Z.

Function:

When this sentence is executed, two type-ins will be set up on the supervisory control printer for each of the listed variables. The operator is expected to type-in the values of the listed variables in two word floating decimal form. See Chapter IV on "Formats" for a discussion of two word floating decimal form.

Rules:

1. Any number of ordinary (not subscripted) variables up to 50 may be listed in a TYPE-IN sentence.

VARY (increment type)

Formats:

(n) VARY X X_0 (X_i) X_L SENTENCES F THRU L.

(n) VARY X X_0 (X_i) X_L Y Y_0 (Y_i) Y_L SENTENCES F THRU L.

(n) VARY X X_0 (X_i) X_L Y Y_0 (Y_i) Y_L Z Z_0 (Z_i) Z_L SENTENCES F THRU L.

Examples:

(27C) VARY X 5 (0.01) 10 SENTENCES 31 THRU 34.

(9) VARY DX 3.17493 (3.1414*10 $^{-8}$) A SENTENCES 17 THRU 20.

(18) VARY Y7 0 (5) 100 Y8 100 (10) 300 Y9 50 (2) 90 SENTENCES 51 THRU 51B.

(3) VARY Y I (10^{-7})|C| SENTENCES 5 THRU 6.

Functions:

The "VARY" sentence of the increment type whose formats are given above creates the mechanism by which sentences F through L are executed repeatedly, each time for different values of X, X and Y, or X, Y and Z. The first time sentences F through L are executed, $X = X_0$, $Y = Y_0$ and $Z = Z_0$. Each time sentences F through L are repeated, X_i is added to X, Y_i is added to Y and Z_i is added to Z. Sentences F through L are repeated until the values of the variables exceed their limits, i.e., until $X > X_L$, $Y > Y_L$, and $Z > Z_L$. Then control passes to the sentence after L.

When sentence n is executed, the variables are given their initial values; X is set equal to X_0, and if Y and Z are also being varied, Y is set equal to Y_0 and Z equal to Z_0. Control then goes to the next sentence after n. After sentence L is executed, each variable is incremented; X is set equal to $X+X_i$, and if Y and Z are being varied, Y is set equal to $Y+Y_i$ and Z is set equal to $Z+Z_i$. The variable X is then tested against its limit X_L. If more than one variable is being varied, the variable named last in the sentence is tested against its limit. If $X \leq X_L$, control is transferred back to sentence F so that sentences F through L may be repeated. If $X > X_L$, control moves to the next sentence after L. If X_i is negative, or if X_L is less than X_0, the above inequalities are reversed; that is, when $X \geq X_L$, control is returned to sentence F and when $X < X_L$, control passes to the next sentence after L.

Conventions:

1. The initial value X_0, increment X_i, and limit X_L, of the variable X may themselves be any previously defined variable with or without absolute value signs, or any integer, decimal or power of ten form.

Rules:

1. No more than three variables may be varied by a VARY sentence.

2. Sentences n, F and L may be in any order in the pseudo-code but in order of execution L must come after F and n must be outside the range F to L.

3. A VARY sentence and its range of execution sentences F through L may be executed any number of times. The range of one VARY sentence may be entirely included in the range of another vary sentence, but two ranges of execution should not overlap each other.

VARY (list type)

Formats:

(n) VARY X X_0 X_1 X_2... X_n SENTENCES F THRU L.

(n) VARY X Y Z X_0 Y_0 Z_0 X_1 Y_1 Z_1 X_2 Y_2 Z_2.... X_n Y_n Z_n SENTENCES F THRU L.

Examples:

(3) VARY X 1 1.1 1.3 1.6 1.85 1.95 2.2 2.7 4.1 SENTENCES 7 THRU 12.

(21) VARY P Q R -4 -1 2 -1 3 7 5 1 9 SENTENCES 25 THRU 25 E.

Functions:

The VARY sentence of the list type sets up the mechanism by which sentences F through L are executed repeatedly, each time for a different set of values of the variables X, Y, and Z. After the last set of values, X_n, Y_n, and Z_n, has been used, control passes to the sentence after L.

When sentence n is executed, X, Y, and Z are given their initial values X_0, Y_0 and Z_0. After sentence L is executed, the next set of values is moved into the positions of the variables and control returns to sentence F. When the sets of values are exhausted, control moves to the next sentence after L.

Rules:

1. The values of the variables, X_0, Y_0, Z_0, X_1, Y_1, etc. must be numbers, not variables. The values may be in integer, decimal or power of ten form.

2. One, two, or three variables may be varied by a VARY sentence of the list type.

3. A maximum of 50 values of all the variables may be listed.

4. It is the user's responsibility to have an integral number of sets of values listed, i.e., the number of values must be evenly divisible by the number of variables.

5. Sentences n, F, and L may appear in any order in the problem statement, but in order of execution sentence n must precede sentence F and sentence F must precede sentence L.

IGNORE

Example:

(10) IGNORE.

Function:

This sentence is not executed during the problem run. It is used in conjunction with a VARY sentence as an artificial last sentence of the range of execution of the VARY sentence. It is needed only when it is desired to transfer control to a point beyond the last functioning sentence of the range but before the incrementation of the VARY sentence. Sample problem 3 of Chapter I illustrates the use of an IGNORE statement.

STOP

Example:

(27B) STOP.

Function:

This sentence must be the last sentence of the MATH-MATIC pseudo-code. When this sentence is executed, MATH-MATIC places an appropriate sentinel on each output tape, and prints out the number of blocks of output on each tape. Every servo used in the problem is rewound and "END OF RUNΔΔ" is printed out. The computer stops.

Rules:

1. The STOP sentence must be the last sentence in the pseudo-code. Only the special COMPUTER and COMPILER sections, if any are used, may follow the STOP sentence.

INPUT-OUTPUT SENTENCES

READ

Formats:

(n) READ A B C.

(n) READ A B C IF SENTINEL JUMP TO SENTENCE F.

(n) READ A B C IF SENTINEL RESET AND JUMP TO SENTENCE F.

(n) READ A B C IF SENTINEL REWIND AND JUMP TO SENTENCE F.

Example:

(5) READ A B C SERVO 3 STORAGE A IF SENTINEL REWIND AND JUMP TO SENTENCE 20.

Functions:

This sentence will assign a new set of values to the listed variables each time it is executed. These sets of values, or input items, must be on tape. The user must read Chapter IV for instructions on how to prepare his input data. During compilation a print-out will tell the user which servo to mount the input tape on; if a servo has been specified, such as SERVOΔ3 in the example, this specified servo number will be printed out. Each of the three "IFΔSENTINELΔ" options means that every time the sentence is executed, the first word of the new input item is compared with a word of Z..Z's the usual UNIVAC sentinel. If the first word does not equal Z..Z's, control moves to the next sentence after (n). If the first word does equal Z..Z's, what occurs before control goes to sentence F depends on which of the three options was employed. With "RESETΔANDΔJUMP" MATH-MATIC expects to find further input on the same tape, and the next time this READ sentence is executed, a new set of values for the listed variables will be read into the memory starting with the next block on the tape. With "REWINDΔANDΔJUMP" MATH-MATIC expects the same input, whose sentinel has just been reached, to be used again. Accordingly, the servo is rewound, and the next time this READ sentence is executed, the first set of values will be read in again. With "JUMP" MATH-MATIC does not expect the READ sentence to be executed again and therefore takes no action. However, a subsequent READ or READ-ITEM sentence specifying the same servo will automatically start reading values from the next block on the tape, or from the first block if the servo has been rewound. In all three "IF∆SENTINEL∆..." cases, when the sentinel is reached, control is transferred to sentence F.

Rules:

1. Sentence F may come before or after sentence n, but it must appear somewhere in the pseudo-code.

2. A list of from one up to 50 variables may appear in a READ sentence. No subscript arrays may be used with a READ sentence.

3. If servo or storage is specified, the words must appear after the list of variables.

4. If one of the "IF∆SENTINEL∆..." options is used, the words must appear at the end of the sentence.

5. Two READ sentences having any variables in common must specify the same storage area and the common variables must occupy the same relative position in the two areas. READ A X B STORAGE A and READ A X C STORAGE A is permissible; READ A B C and READ D A F, regardless of storage specification, is not permissible.

READ-ARRAY

Format:

(n) READ-ARRAY X(m, n).

Examples:

(1) READ-ARRAY X(10, 10).

(6) READ-ARRAY X(2, 3,.4) SERVO 4 STORAGE B.

Functions:

Regardless of where this sentence appears in pseudo-code, it will be executed before the running program begins. It will read from tape into memory, the values of the number of elements implied by the subscript array. In the first example there are 10x10 = 100 elements of 200 words; in the second example there are 24 elements or 48 words implied by the array. The user must read Chapter IV for instructions on how to prepare his input data, and how to arrange the elements within an array. During compilation MATH-MATIC will print out the servo reserved for the array; if a servo is specified in the pseudo-code, as SERVO∆4 is in the second example, this servo number will be printed out.

Rules:

1. Each READ-ARRAY sentence may be executed only once.

2. Any subscripted array with one, two, or three subscripts, implying from one to 250 elements, may appear in a READ-ARRAY sentence. A list of variables may not appear in a READ-ARRAY sentence.

3. If servo is specified, the words must appear after the subscript array.

4. No "IF∆SENTINEL∆" options are allowed with a READ-ARRAY sentence.

5. A maximum of seven READ-ARRAY's in one problem statement is allowed.

6. The complete name of the subscript array, including the parentheses and commas, must not exceed 12 digits in length.

READ-ITEM

Formats:

(n) READ-ITEM X(m, n).

(n) READ-ITEM X(p) IF SENTINEL JUMP TO SENTENCE F.

(n) READ-ITEM ALPHA (m, n, p) IF SENTINEL RESET AND JUMP TO SENTENCE F.

(n) READ-ITEM B(m, n) IF SENTINEL REWIND AND JUMP TO SENTENCE F.

Example:

(22A) READ-ITEM MATRIX(3, 12) SERVO 8 STORAGE E IF SENTINEL REWIND AND JUMP TO SENTENCE 60.

Functions:

Each time this sentence is executed, it will assign a new set of values to the elements of the subscript array. These sets of values, or input items, must be on tape. The user must read Chapter IV for instructions on how to prepare his input data, and on how to arrange the elements within an array. During compilation, a printout tells the user on which servo to mount the input tape; if a servo is specified, as in the example, the specified servo number is printed out. Each of the three "IF△SENTINEL△..." options means that every time the sentence is executed, the first word of the new input item is compared with a word of Z..Z's, the usual UNIVAC sentinel. If the first word does not equal Z..Z's, control moves to the sentence following n. IF the first word does equal Z..Z's, what occurs before control goes to sentence F depends on which of the three options was employed. With "RESET△AND△JUMP" MATH-MATIC expects to find further input on the same tape, and the next time this READ-ITEM sentence is executed, a new set of values for the elements of the array will be read into memory starting from the next block on the tape. With "REWIND AND JUMP"... MATH-MATIC expects the same input, whose sentinel has just been reached, to be used again. Accordingly the servo is rewound, and the next time this READ-ITEM sentence is executed, the first set of values for the elements of the array will be read in again. With "JUMP","MATH-MATIC does not expect this READ-ITEM sentence to be executed again and does nothing. However, a subsequent READ or READ-ITEM sentence specifying the same servo will automatically start reading values from the next block on the tape, or from the first block if the servo has been rewound. In all three "IF△SENTINEL"...cases control is transferred to sentence F.

Rules:

1. Sentence F may come before or after sentence n, but sentence F must appear somewhere in the pseudo-code.

2. If servo or storage is specified, the words must appear after the name of the subscript array.

3. The complete name of the sub-

script array, including the parentheses and commas, must not exceed 12 digits in length.

4. Any subscripted array with one, two, or three subscripts implying from one to 250 elements, may appear in READ-ITEM sentence. A list of variables may not appear in a READ-ITEM sentence.

WRITE

Formats:

(n) WRITE X Y Z.

(n) WRITE FOR UNIPRINTER X Y Z.

(n) WRITE EDIT X Y Z.

(n) WRITE EDIT FOR UNIPRINTER X Y Z.

(n) WRITE CONVERT X Y Z.

Examples:

(12) WRITE CONVERT A B X Y N SERVO 5 STORAGE G.

(5B) WRITE ALPHA SERVO 9.

Functions:

Each time the WRITE sentence is executed, the current values of the listed variables are written on tape. The writing is for the high speed printer unless the "FOR△UNIPRINTER" option is used. The values of the variables are in two word floating decimal form unless "EDIT" or "CONVERT" has been specified in the sentence. "EDIT" and "CONVERT" cause the values of the variable to be edited into special forms before being written on tape. During compilation, MATH-MATIC prints out the number of the servo assigned to this output sentence. If a servo has been specifed in the sentence, the specified servo number is printed out. MATH-MATIC arranges to place sentinels on the output tape when the problem run is completed. The information on the tape is checked for readability and the number of output blocks on the tape is printed out. The sentinel operation fills the last block on the tape with Z...Z's for unedited output and with printers stops (Σ's) for edited output. The last section of Chapter IV discusses output item sizes and edited forms.

Rules:

1. Any number of variables up to 50 may be listed in a WRITE sentence. A subscript array may not be used in a WRITE sentence.

2. If servo or storage is specified, the specification must appear after the list of variables.

WRITE-ITEM

Formats:

(n) WRITE-ITEM X(m, n, p).

(n) WRITE-ITEM FOR UNIPRINTER X(m, n, p).

(n) WRITE-ITEM EDIT X(m, n, p).

(n) WRITE-ITEM CONVERT X(m, n, p).

(n) WRITE-ITEM EDIT FOR UNIPRINTER X(m, n, p).

Examples:

(17) WRITE-ITEM A(8, 9) SERVO 6.

(23B) WRITE-ITEM EDIT Z(5, 5, 5) STORAGE F SERVO 3.

Functions:

Each time the WRITE-ITEM sentence is executed, the current values of the elements of the subscript array are written on tape. The writing is for high speed printer unless the "FORΔ UNIPRINTER" option is used. The values of the elements are in two word floating decimal form unless "EDIT" or "CONVERT" has been specified in the sentence. "EDIT" and "CONVERT" cause the values of the elements to be edited into special forms before being written on tape. During compilation, MATH-MATIC prints out the number of the servo assigned to this output sentence. If a servo has been specified in the sentence, the specified servo number is printed out.

MATH-MATIC arranges to place sentinels on the output tape when the problem run is completed. The information on the tape is checked for readability and the number of output blocks on the tape is printed out. The sentinel operation fills the last block on the tape with Z...Z's for unedited output and with printer's stops (Σ 's) for edited output. Output item sizes, edited forms, and the arrangement of elements within an array are all discussed in Chapter IV.

Rules:

1. Any subscripted array with one, two, or three subscripts, implying from one to 250 elements, may appear in a WRITE-ITEM sentence. A list of variables may not appear in a WRITE-ITEM sentence.

2. The complete name of the subscript array, including the parentheses and commas, must not exceed 12 digits in length.

3. If servo or storage is specified, the specification must appear after the name of the subscript array.

WRITE-ARRAY

Formats:

(n) WRITE-ARRAY X(m, n, p).

(n) WRITE-ARRAY FOR UNIPRINTER X(m, n, p).

(n) WRITE-ARRAY EDIT X(m, n, p).

(n) WRITE-ARRAY CONVERT X(m, n, p).

(n) WRITE-ARRAY EDIT FOR UNIPRINTER X(m, n, p).

(n) WRITE-ARRAY EDIT AND CONTAIN X(m, n, p).

Examples:

(8) WRITE-ARRAY A(10,12) STORAGE A SERVO 6.

(16) WRITE-ARRAY CONVERTED RT(25) SERVO 2.

Functions:

Each time the WRITE-ARRAY sentence is executed, the current values of the elements of the subscript array are written on tape. The writing is for the high speed printer unless the "FORΔUNIPRINTER" option is used. The values of the elements are in two word floating decimal form unless "EDIT" or "CONVERT" has been specified in the sentence. "EDIT" and "CONVERT" cause the values of the elements to be edited into special forms before being written on tape. During compilation, MATH-MATIC prints out the number of the servo assigned to this output sentence. If a servo has been specified in the sentence, the specified servo number is printed out.

MATH-MATIC arranges to place sentinels on the output tape when the problem run is completed. The information on the tape is checked for readability and the number of output blocks on the tape is printed out. The sentinel operation fills the last block on the tape with Z. . . Z's for unedited output and with printer's stops (Σ's) for edited output. Output item sizes, (particularly when WRITE-ARRAY differs from WRITE-ITEM), edited forms, and arrangements of elements within an array are all discussed in Chapter IV. If the user desires to perform further calculations on elements which have been edited and written, he must use a WRITE-ITEM EDIT sentence or a WRITE-ARRAY EDIT AND CONTAIN sentence. If the user does not wish to perform further calculations on the elements, a WRITE-ARRAY EDIT sentence is permissible and will save storage space.

Rules:

1. Any subscripted array with one, two, or three subscripts, implying from one to 250 elements, may appear in a WRITE-ARRAY sentence. A list of variables may not appear in a WRITE-ARRAY sentence.

2. The complete name of the subscript array, including the parentheses and commas, must not exceed 12 digits in length.

3. If servo or storage is specified, the specification must appear after the name of the subscript array.

4. Further calculations may not be performed on elements which have been edited and written in a WRITE-ARRAY EDIT or WRITE-ARRAY CONVERT sentence.

CLOSE-INPUT

Formats:

(n) CLOSE-INPUT SENTENCE F.

(n) CLOSE-INPUT **AND** REWIND SENTENCE F.

Example:

(6) CLOSE-INPUT AND REWIND SENTENCE 2.

Function:

When the "CLOSE-INPUT" sentence with no "REWIND" is executed, MATH-MATIC expects further input on the servo assigned to the input sentence F. The next time input sentence F is executed, values will be read into the memory starting with the next block on the tape.
When the "CLOSE-INPUT AND REWINDΔ" sentence is executed, MATH-MATIC expects the input of sentence F to be used again. Accordingly, the servo assigned to sentence F is rewound and, the next time sentence F or another input sentence with the same servo is executed, values will be read into memory starting with the first block on the tape.

Rules:

1. Sentence F may come before or after sentence n, but it must be an input sentence.

CLOSE-OUTPUT

Formats:

(n) CLOSE-OUTPUT SENTENCE F.

(n) CLOSE-OUTPUT AND REWIND SENTENCE F.

Example:

(17A) CLOSE-OUTPUT AND REWIND SENTENCE 11.

Functions:

When a "CLOSE-OUTPUT" sentence with no "REWIND" is executed, a sentinel item filled with Z... Z's is placed on the output tape of sentence F. MATH-MATIC then prints out the number of blocks of output, written by sentence F. A later output sentence may now add data to the same tape.

When a "CLOSE-OUTPUT AND REWIND" sentence is executed, a sentinel item filled with Z... Z's is placed on the output tape of sentence F, and the output servo is rewound. MATH-MATIC checks the readability of the information written by sentence F, and prints out the number of blocks of output written by sentence F. The data on this servo may now be used as input by a later READ or READ-ITEM sentence. Any of the three "IF SENTINEL..." options may be included in this later READ or READ-ITEM sentence since a sentinel item of Z... Z's was placed on the output (now the input) tape.

Rules:

1. Sentence F may come before or after sentence n in the pseudo-code, but sentence F must be an output sentence.

2. If the user desires to use an output later as an input, the original output sentence must be a WRITE or a WRITE-ITEM sentence, not a WRITE-ARRAY sentence, and it must be unedited.

Note: When using MATH-MATIC with UNIVAC II, the limit on input and output subscript arrays is increased from 250 to 800 quantities.

IV

DATA PREPARATION AND FORMATS

I. DATA PREPARATION:

The MATH-MATIC system uses a two word floating decimal mode of computation. The user may write the input data for the READ, READ-ITEM and READ-ARRAY sentences directly in the two word floating decimal form, or he may prepare his input data in a simpler form and use the Data Conversion routine to convert the data into the two word floating decimal form. For the purpose of input data preparation, the elements of an array in a 'READ-ITEM' of a 'READ-ARRAY' sentence may be considered input variables, and an input array of n elements may be considered an n-variable input item.

There are three forms in which the values of input variables may be prepared before entering the Data Conversion routine: integer, decimal, and power of ten form. Except for one special case of the power of ten form, each value occupies one UNIVAC word space-filled to the right.

Some examples of integer input values are:

5ΔΔΔΔΔΔΔΔΔΔΔ

163ΔΔΔΔΔΔΔΔΔ

81515ΔΔΔΔΔΔΔ

-36ΔΔΔΔΔΔΔΔΔ

2100000000ΔΔ

+6893ΔΔΔΔΔΔΔ

0ΔΔΔΔΔΔΔΔΔΔΔ

Some examples of decimal input values are:

17.8ΔΔΔΔΔΔΔΔ

-.00384ΔΔΔΔΔ

+160.249ΔΔΔΔ

.00552ΔΔΔΔΔΔ

0.1663ΔΔΔΔΔΔ

129546.87312

A power of ten form value may occupy one or two UNIVAC words. If two words are used, the first word contains the significant digits and the second word contains the ten and its exponent. The exponent must be a positive or negative integer. The first digit of the second word must be a multiplication sign (*). Some examples of power of ten form are;

278*10^{3}ΔΔΔΔ

-91.45*10^{6}ΔΔ

+.134*10^{-12}Δ

64831.259ΔΔΔ
*10^{11}ΔΔΔΔΔΔΔ } 2 word power of ten form

-3.5917266ΔΔ
*10^{-8}ΔΔΔΔΔΔΔ } 2 word power of ten form

The following rules apply to all three forms:

1. No more than 11 significant digits may be written in one UNIVAC word.
2. A plus sign may be placed in the first digit of the UNIVAC word containing the significant digits. The plus sign, however, is not necessary.
3. No spaces may be written in front of or between the significant digits, the decimal point, the ten and its exponent, and any +, - or * signs.
4. Spaces must fill each word to the right; if the digits and symbols fill the UNIVAC word, space-filling is not needed.

Each set of data is prepared one input item followed directly by another. No space is left between the items, and items may be divided between blocks. Two items in one set of data must contain the same number of input values but are not necessarily written in the same number of UNIVAC words, because of the variable size power of ten form.

Each set of data must have a sentinel word of Z..Z's in the word following the last valid item. Each additional set of data on the same tape must begin in new block.

The user may prepare his sets of data on different tapes. The converted data may be written all on one tape, or on as many tapes as is desired.

The user is cautioned that the process of converting input data may expand the data to as much as 4 times its original length on tape. Since a full UNIVAC tape holds approximately 2000 blocks (at high density), the user should not convert more prepared data blocks on to one tape than the tape can hold.

If the input sentence is a READ sentence, calling for a list of variables, the values in each prepared input item should be in the same order as the variables. For example, if the input sentence says READΔAΔXΔB, each input item consists of 3 values; the first value represents A, the second value X, and the third value B.

II. Arrangement of elements within an array

This discussion applies to all subscript arrays whether input, unedited output or edited output. Item padding, overlay blocks, and forms of the numbers depend on the pseudo-code sentence and are all discussed elsewhere in this chapter, but the elements are always arranged within the array in the same order.

In a subscript array with one subscript the elements are arranged in ascending order. In any reference to a subscripted variable with one subscript the subscript denotes the position of the element in the array; i.e., X(15) is the 15th element.

In a subscript array with two subscripts, the values of the elements are arranged one complete row after another. In any reference to a subscripted variable with two subscripts, the first subscript represents the row, and the second subscript represents the column in the array.

In a subscript array with three subscripts, the values of the elements are arranged one complete plane after another, and within each plane, one complete row after another. In any reference to a subscripted variable with three subscripts the first subscript represents the plane, the second subscript represents the row, and the third subscript represents the column in the array.

In the case of the array X(20), there are 20 elements arranged as follows; X(1), X(2), X(3), . . . , X(20).

In the case of the array B(8, 7), there are 8 rows and 7 columns, or 56 elements running from B(1, 1) to B(8, 7). The 56 values are arranged in the following order: B(1, 1), B(1, 2), ... B(1, 7); B(2, 1), B(2, 2)... B(2, 7) B(3, 1)... B(8, 1)... B(8, 7).

In the case of the array ALPHA(3, 4, 5) there are 3 planes, 4 rows, and 5 columns, or 60 elements running from ALPHA(1, 1, 1) to ALPHA(3, 4, 5). The 60 values are arranged within the total array as follows: ALPHA(1, 1, 1), ALPHA(1, 1, 2)... ALPHA(1, 1, 5); ALPHA(1, 2, 1)... ALPHA(1, 2, 5), ALPHA(1, 3, 1)... ALPHA(2, 1, 1)... ALPHA(3, 1, 1)... ALPHA(3, 4, 5).

III. Use of the Data Conversion Routine

The Data Conversion routine takes input in integer, decimal, and power of ten form and converts it to the two word floating decimal form MATH-MATIC requires for its internal operations. The Data Conversion routine also arranges the input items for the system by padding the incomplete item sizes, establishing overlay block in the large input items, and providing sentinels for each set of input data. If the user does not want to make use of the Data Conversion routine, he must prepare his data in the two word floating decimal form and he must include the necessary padding, overlays and sentinels himself.

The user may convert his data any time prior to the problem run. For convenience, the Data Conversion routine will appear at the beginning of every compiled program, as well as being separately available. Thus the user may convert and properly distribute his data immediately before running. The control words for the data conversion routine may be typed in at the supervisory control at the time of conversion, or may be prepared on tape. In all cases, the routine prints out the number of blocks in each set of converted input data, checks the readability of each converted tape and rewinds all tapes.

To convert data independently of the problem run mount the Data Conversion routine on any servo t, and depress servo selector key t. Initial read servo t. Immediately upon starting the program the follow print-out occurs.

```
CONTROLΔTAPE
```

A type-in is set up. If the tape control option is being used, type in:

```
XXXXXXXXXXXX
```

where X is the number of the servo containing the control tape.

The Data Conversion routine converts all the data and goes into the Conversion Ending routine.

If the type-in control option is being used, type in

```
SCPΔCONTROLΔ
```

These print-outs and type-ins follow:

PRINT-OUT	TYPE-IN
READΔSENTNCE	READΔΔΔΔΔΔΔΔ or READ-ARRAYΔΔ or READ-ITEMΔΔΔ; ZZZZZZZZZZZZ if there are no further data.
ITEMΔSIZEΔΔΔ	XΔΔΔΔΔΔΔΔΔΔΔ where X is the number of variables or elements in each input item; if X > 9, type in XXΔΔΔΔΔΔΔΔΔΔ or XXXΔΔΔΔΔΔΔΔΔ.
SERVOSΔΔΔΔΔΔ	AAAAAABBBBBB where A is the servo containing the prepared data and B is the servo desired for the converted data; A ≠ B.
XXXXΔBLOCKSΔ ONΔSERVOΔPΔΔ	No type-in called for; the routine has finished converting the set of data and is ready for another one.

One set of these printouts and type-ins takes place for each set of input data. The printouts and type-ins continue until a word of Z... Z's is typed in after READΔSENTNCE has printed out. Then the Conversion Ending routine is entered.

If more than one set of prepared input data is on one tape, the same servo A will be typed in more than once. If the user desires to put more than one set of converted data on one tape, the same servo B will be typed in more than once.

If tape control is used, the first block on the control tape must be in the form:

HEADERΔΔΔΔΔΔ	3 word header item; may be anything	
HEADERΔΔΔΔΔΔ		
HEADERΔΔΔΔΔΔ		
READΔΔΔΔΔΔΔΔ	as many of these groups are there are sets of data (up to 18)	Type of read sentence
XΔΔΔΔΔΔΔΔΔΔΔ		item size
AAAAAABBBBBB		Servo A for prepared data Servo B for converted data.

```
. . . . . . . . . . . . )
. . . . . . . . . . . .  }    last group
AAAAAABBBBBB )

ZZZZZZZZZZZZ       sentinel

Δ. . . . . . . . . . Δ     insignificant words for the rest of the block.
```

The first 3 words in the block may be any header the user wishes. The three word groups are the same as the three type-ins described above. The following printouts occur each time a three word group is processed:

XXXXΔBLOCKSΔ

ONΔSERVOΔBΔΔ

The Conversion Ending routine functions automatically after the last set of data has been converted. Every tape which has had converted data written on it is checked for readability and all tapes are rewound. The Computer stops after "ENDΔCONVERTΔ" is printed out.

To run the Data Conversion routine on the compiled running tape, mount the running tape on any servo, t, and depress the servo selector key for servo t. Set break point #1 and initial read servo t. Force transfer on break point #1 and the following printout occurs:

CONTROLΔTAPE

If transfer is not forced, the problem run will be started immediately. From this point the options printouts and type-ins are the same as when the Data Conversion routine is run separately.

Again, after the Conversion Ending routine takes place, "ENDΔCONVERTΔ" print out and the computer stops, leaving the running tape positioned for the problem run. All other servos are rewound. At this point the user need remove only those tapes which should be blank in order to receive output during the problem run. The user should select the servo B's of the control words so that each set of converted data is placed on the servo specified for it in the pseudo-code or in the compilation. Then those tapes may remain where they are at the end of the Conversion Ending routine. After the servo set-up is checked for conformity with the compilation printouts, the start bar should be hit to begin the problem run.

IV. Formats

The MATH-MATIC system expects all values which are read in or used in calculation during the problem run to be in two word floating decimal form:

$\underline{0}$XXXXXXXXXXX 11 significant digits

$\underline{0}$000000000EEE ten's exponent

This form represents the number $\pm$.XXXXXXXXXXX * $10^{\pm EEE}$. The first digit after the sign in the first word must be non-zero, unless the value itself is zero. Some examples of two word floating decimal form follow:

$$\left.\begin{array}{l}036100000000\\000000000002\end{array}\right\} = 36.1$$

$$\left.\begin{array}{l}017487000000\\-00000000003\end{array}\right\} = .00017487$$

$$\left.\begin{array}{l}-84666137191\\000000000002\end{array}\right\} = -84.666137191$$

$$\left.\begin{array}{l}000000000000\\000000000000\end{array}\right\} = 0$$

$$\left.\begin{array}{l}-15400000000\\000000000011\end{array}\right\} = -15,400,000,000$$

Unedited output is also in two word floating decimal form.

All inputs and outputs handled in READ, READ-ITEM, unedited WRITE and unedited WRITE-ITEM sentences are arranged in allowable item sizes by means of padding and block overlays. If the Data Conversion routine is bypassed, the user must arrange the padding and block overlays in his input data. In all other cases MATH-MATIC arranges the items according to the following rules;

1) The numbers of variables which make up allowable item sizes are 1, 2, 3, 4, 5, 6, 10, 15, 30, 35, 40, 45, 50, 55 60, 65, 70, 75, 80, ..., 235, 240, 245 and 250. If the number of variables in the actual item is not one of the allowable item size, the item is padded at the end with the number of dummy variables needed to make it an allowable item size; e. g., a 7 variable item will have 3 dummy variables at the end to make up a 10 variable item. Each dummy variable consists of 2 words of zeroes since the values are expected to be in two word floating decimal form. A 74 variable item will be padded at the end with one dummy variable (2 words) to make up a 75 variable item, which is an allowable item size.

2) If the number of variables in the item is 30 or less, the following chart gives the distribution of items per block;

No. of variables	No. of words	No. of items per block
1	2	30
2	4	15
3	6	10
4	8	7
5	10	6
6	12	5
10	20	3
15	30	2
30	60	1

In the 4 variable (8 word) case the 7 items occupy the first 56 words of the block, leaving 4 insignificant words at the end of the block. In the other cases on the above chart the items fill each block entirely.

3) If the number of variables in the item is greater than 30, each item will occupy more than one block. To determine the number of blocks occupied by each item, divide the number of variables, n, by 30. Call the quotient of the division q and the remainder r. For example, if n = 85, q = 2 and r = 25. If n is evenly divisible by 30, (r=0), the item entirely fills q blocks and no overlay is used. If $r \neq 0$, the item occupies q+1 blocks. Every block in the item except the next to last block is filled with 30 va-

lues of variables. The next to last block contains r values of variables at the beginning of the block and insignificant words in the remainder of the block. When the items are read into the memory, the last block will overlay the insignificant words of the next to last block. For example, an 85 variable item would be arranged in 3 blocks as follows; the first 30 variables in the first block, the next 25 variables in the second block, and the last 30 variables in the third and last block. The third block will overlay the last 10 words of the second block.

For READ-ARRAY and unedited WRITE-ARRAY sentence the values of the variables are in two word floating decimal form. The values of the one set of data for a READ-ARRAY sentence are arranged consecutively on tape without padding or overlays. The first 30 values in the set of data fill the first block. Any further values start from the beginning of the second block. Any values in excess of 60 start in the third block, and so forth. The values in each output item of an edited or an unedited WRITE-ARRAY sentence are arranged consecutively on tape without overlays. Each of them occupies an integral number of blocks on tape. Any item size from one up to 30 variables is padded to 30 variables to fill one block. Any item size from 31 to 60 variables is padded to 60 variables to fill two blocks, and so forth. If the item size is multiple of 30, such as 30, 60, 90, 120 etc., no padding is needed. In an unedited WRITE-ARRAY sentence the padding consists of irrelevant values from another storage area. The padding in an edited WRITE-ARRAY item occupying one block, also consists of irrelevant numbers, but if the item occupies more than a block, the padding consists of spaces.

The values of the variables in each item of an edited WRITE or edited WRITE-ITEM sentence are written consecutively without overlays. There will be padding at the end of each item if the number of variables is not an allowable edited item size. The padding consists of spaces regardless of item size. The allowable edited item sizes are 5, 10, 15, 30, 60, 90, 120, 150, 180, 210, 240, and 270. The number of items per block for the item sizes of 30 or less are given in the following chart:

No. of variables	No. of words	No. of items per block
5	10	6
10	20	3
15	30	2
30	60	1

Each item size greater than 30 occupies the integral number of blocks required at 30 values per block. For example, a 130 variable item is given 20 variables of padding (40 words of spaces) to make an allowable 150 variable item which will occupy 5 blocks. The user is reminded of the 50 variable limit in a list of variables and the 250 variable limit in a subscript array.

If it is desired to perform further calculations with values of variables which have been edited a WRITE-EDIT, WRITE-ITEM EDIT or WRITE-ARRAY EDIT AND CONTAIN sentence must be used. A WRITE-ARRAY EDIT sentence may not be used if further calculations are desired. If further calculations are not desired, a WRITE-ARRAY EDIT sentence is recommended for efficiency in use of storage space.

For WRITE EDIT FOR UNIPRINTER and WRITE-ITEM EDIT FOR UNIPRINTER sentences, there are five additional allowable edited item sizes 1, 2, 3, 5 and 6. The following chart gives the number of edited items per output block;

No. of variables	No. of words	No. of items per block
1	2	30
2	4	15
3	6	10
4	8	7
6	12	5

In the case of a 4 variable item, the 7 items occupy the first 56 words of each block, leaving four words of spaces at the end of each block. In each of the other cases the block is filled with edited values.

The "EDIT" form is:

$\pm$. XXXXXXXXXXX(+EE)

This form represents the number $\pm$. XXXXXXXXXXX * $10^{\pm EE}$. Some examples of the "EDIT" form are:

+.16543397800(+3)

-.81654000000(+0)

+.50041913644(-10)

If EE > 99 in the floating decimal value, an error printout occurs and the value is transferred to output storage unmodified.

The "CONVERT" form is;

$\pm$XXXX. XXXXXXX(if EE $\geq$0) or

$\pm$. 000XXXXXXXXXXX(if EE < 0)

Some examples of the "CONVERT" form are:

+75.163000000

-.26499300000

+.00000817143000

If EE > 10 in the floating decimal value, an error printout occurs and the value is transferred to output storage unmodified.

If non-numeric words are picked up by either the EDIT or the CONVERT routines, the words will be transferred to output storage unmodified.

Note: When using MATH-MATIC with UNIVAC II, the limit on input and output subscript arrays is increased from 250 quantities to 800 quantities. Also the maximum input-output storage is increased from 660 words to 1700 words.

V

COMPUTER AND COMPILER SECTIONS

I Purpose

Occasionally a problem will go beyond the present MATH-MATIC repertoire. The user can write COMPUTER or COMPILER sentences in his pseudo-code to handle these special problems. Each COMPILER sentence calls for the insertion of a number of ARITHMATIC operations into the problem statement. Each COMPUTER sentence calls for the insertion of a number of UNIVAC instructions into the problem statement. The sets of inserted instructions are called the COMPUTER or COMPILER sections. For each COMPUTER or COMPILER sentence in the pseudo-code there must be a corresponding COMPUTER or COMPILER section. These sections of UNIVAC instructions and ARITH-MATIC operations are prepared on the pseudo-code tape following the STOP sentence, and after the DIRECTORY, if one is needed. Each COMPUTER or COMPILER section must begin in a new block and may be any number of blocks in length. A sentinel block with Z. . . Z's in the first and last words must follow the last COMPUTER or COMPILER section. The DIRECTORY is a list of constants and variables referred to in the COMPUTER and COMPILER sections. The DIRECTORY begins the next block after the STOP sentence and precedes the COMPUTER and COMPILER sections. The next part of this chapter gives the rules and conventions of COMPUTER and COMPILER sentences. The rest of this chapter gives rules and examples of the DIRECTORY, COMPUTER Sections, and COMPILER Sections. The repertoire of ARITH-MATIC operations appears in the appendix of this manual.

II COMPUTER and COMPILER Sentences

A. FORMAT

(n) COMPUTER-XXX.

(n) COMPUTER-XXX STORAGE Y (ttt) STORAGE Y(ttt) SERVO s.

(n) COMPILER-XXX.

(n) COMPILER-XXX STORAGE Y(ttt) SERVO s STORAGE Y(ttt).

where XXX is the COMPUTER or COMPILER label, Y the storage area designation, ttt the storage area size, and s the servo number.

B. EXAMPLES

(3) COMPUTER-3B STORAGE A(60) SERVO 7 STORAGE B(10) STORAGE C(60) SERVO - .

(46) COMPILER-SB7 STORAGE I(60) SERVO 3.

C. FUNCTION

A COMPUTER or COMPILER sentence informs MATH-MATIC that a COMPUTER or COMPILER section appears after the STOP sentence. The machine code produced by the system for these sections will be inserted in the running program for execution as sentence (n). Storage areas and associated servos, if needed in the COMPUTER or COMPILER sections, must be specified in the sentences.

D. CONVENTIONS

1. For each COMPUTER or COMPILER section there must be a corresponding COMPUTER or COMPILER sentence.

2. The label (XXX) of the COMPUTER or COMPILER sentence must match the label of the header of the COMPUTER or COMPILER Section.

3. The storage area (Y), if one is used, is designated by an alphabetic character other than W.

4. The storage area size (ttt) is the number of UNIVAC words needed for storage. (ttt) may be any even number up to 500.*

5. The servo number(s) may be any available servo from 1 through 9 and - .

6. The label (XXX) may be numeric, alphabetic, or a combination of numerics and alphabetics. The label may be one, two, or three digits in length.

7. If a read or write order appears in the COMPUTER or COMPILER section, the storage area and the servo must be specified in the corresponding sentence.

8. Any amount of English description may be used in these sentences to describe the function of the COMPUTER and COMPILER sections. This description follows the storage area and servo assignment, but comes before the space period (Δ.) which ends the sentence. In writing this description, care must be taken not to use words of more than 12 digits.

III The DIRECTORY

The DIRECTORY is a list of constants and variables referred to in COMPUTER and COMPILER sections. Any variable, which the user wishes to refer to in a COMPUTER or COMPILER section and which appears elsewhere in the pseudo-code, must be listed in the DIRECTORY. The first entry in the DIRECTORY after the header is referred to by the address "W01" in the COMPUTER and COMPILER sections; the second entry is "W02," and so forth. Each one word entry in the DIRECTORY represents a two word floating decimal quantity.

Example	Address in COMPUTER and COMPILER sections
DIRECTORYΔΔΔ	Header
6ΔΔΔΔΔΔΔΔΔΔΔ	W01
-27.74ΔΔΔΔΔΔ	W02
XΔΔΔΔΔΔΔΔΔΔΔ	W03
YΔΔΔΔΔΔΔΔΔΔΔ	W04
$81*10^{-8}$ΔΔΔΔΔ	W05
ALPHAΔΔΔΔΔΔΔ	W06
ENDΔDIRECTRY	Sentinel

A. CONVENTIONS

1. The DIRECTORY is a means of cross-referencing variables and constants among COMPUTER

*On UNIVAC II, this limit is raised to 1600.

and COMPILER sections and the pseudo-code sentences. One DIRECTORY must suffice for all cross-referencing needs in each problem statement.

2. The DIRECTORY must begin in a new block immediately following the pseudo-code sentences and must precede the COMPUTER and COMPILER sections.

3. The DIRECTORY must start with the header, "DIRECTORY ΔΔΔ," and must finish with the sentinel, "ENDΔDIRECTRY." (Note the missing O). The sentinel must immediately follow the last valid entry in the DIRECTORY.

4. A maximum of 99 entries, corresponding to the address W01 to W99, are permitted in the DIRECTORY. In the above example the last entry, "ALPHA," has the address W06.

5. All entries in the DIRECTORY must begin at the left of the UNIVAC word and must be space-filled to the right.

6. The constants in the DIRECTORY may be integers, decimals, or in power of ten form.

7. A set of consecutive entries in the DIRECTORY are not always given consecutive storage locations by MATH-MATIC, and the user is cautioned against depending on this. A set of consecutive entries in the DIRECTORY will be given consecutive storage locations only if none of the entries in the set appears in an input or output sentence elsewhere in the pseudo-code statement.

The user should regard the DIRECTORY as a cross-referencing device rather than as a storage area.

IV COMPILER section

A COMPILER section consists of ARITH-MATIC operations. The repertoire of ARITH-MATIC operations appears in the appendix of this manual. The operations in the COMPILER section will be executed during the problem run at the point in the pseudo-code where the corresponding COMPILER sentence appears.

<u>EXAMPLE</u>

The COMPILER sentence is given first, then the COMPILER section corresponding to the sentence.

<u>The sentence</u>

(10) COMPILER-ABC SERVO 3 STORAGE D(80).

<u>The section</u>	<u>Operation number for reference in the COMPILER section</u>
COMPILER-ABC	
GMI0 03080D00	M000
TS0 W04000W05	M001
AA0 D16W02W06	M002
ALLD 20D76 W04	M003
01CN000035ΔΔ	
02CN00M002ΔΔ	
AM0 D50D40D30	M004
ENDΔCOMPILER	

CONVENTIONS

1. Each COMPILER section must begin in a new block and must come after the pseudo-code sentences and after the DIRECTORY.

2. The first word of the COMPILER section must be a header which is exactly the same as the first word of the corresponding COMPILER sentence.

3. The COMPILER section may include any number of blocks, and the first word after the last valid operation must be the sentinel, "ENDΔCOMPILER."

4. The symbolic addresses in the ARITH-MATIC operations must be one of the following:

   a) W address referring to entries in the DIRECTORY.

   b) symbolic address referring to the storage area(s) specified anywhere in the pseudo-code. "D16" in operation M002 is an example of this.

5. The operations in a COMPILER section are numbered consecutively relative to the first operation. The first operation is numbered M000. The XXCN's do not take operation numbers. The operation numbers are for user's reference only, and should not appear in the COMPILER section.

6. An "XXCN" with a "0" in the 7th digit position calls for a transfer of control to the pseudo-code sentence whose number appears in the 8th through the 10th digit positions. The 01CN in the example transfers control to sentence 35. If the sentence number has an appended alphabetic, the alphabetic will be in the 11th digit position. For example, a transfer to sentence 7B would be written as XXCN000007BΔ.

7. An "XXCN" with an "M" in the 7th digit position calls for a transfer of control to another operation within the COMPILER section. The operation number to which control is being transferred appears in the 8th through the 10th digits of the "XXCN." The "02CN" in the example transfers control to operation M002 in the COMPILER section, the AA0 operation.

V COMPUTER sections

A COMPUTER section consists of UNIVAC instructions. The instructions in a COMPUTER section will be executed during the problem run at the point in the pseudo-code where the corresponding COMPUTER sentence appears.

EXAMPLE:

The COMPUTER sentence is given first, then the COMPUTER section corresponding to the sentence.

The sentence

(17) COMPUTER-2 SERVO 6 STORAGE B(70).

The section	Line number for reference in the COMPUTER section
COMPUTER-2ΔΔ	

```
V0W001W0B010  M001

B0M001A0M004  M002

A0M001U0M006  M003

CONSTANTS002  The two following
              lines remain un-
              modified.

000000000002  M004 unmodified
                   during
111111111000  M005 compilation

L000STQ0M001  M006

000000T001CN  M007

000000000000  M008

56B010B0B005  M009

H0W103F0W006  M010

G0W00310B009  M011

000000Q002CN  M012

000000000000  M013

B0B004C000ST  M014

ENDΔOWNΔΔΔΔΔ  ARITH-MATIC
              sentinel

STORAGE00001  number of "ST"
              lines reserved

01CN000030ΔΔ

02CN000006AΔ

ENDΔCOMPUTER  MATH-MATIC
              sentinel
```

CONVENTIONS:

1. Each COMPUTER section must begin in a new block and must come after the pseudo-code sentences and after the DIRECTORY.

2. The first word of the COMPUTER section must be a header which is exactly the same as the first word of the COMPUTER sentence.

3. The COMPUTER section may include any number of blocks.

4. The two sentinels, "ENDΔOWN ΔΔΔΔΔ" and "ENDΔCOMPUTER" must immediately follow the last valid instruction of the COMPUTER section. "STORAGE00nnn" and XXCN references, if needed, are placed between the two sentinels.

5. The symbolic addresses in the UNIVAC instructions lie in the 3rd through 6th digit positions and the 9th through 12th digit positions of instruction. Each symbolic address must be one of the following three types;

   a) M in the 3rd or 9th digit position; the reference is to another line in the COMPUTER section. The conditional transfer in line M006 of the example is to line M001.

   b) W in the 3rd or 9th digit position; The reference is to an entry in the DIRECTORY. If "W" is in the 3rd digit position, the 5th and 6th digit positions will contain the DIRECTORY reference. For example, the left half of line M001 refers to W01. The 4th digit position must be "0" or "1." A "0" refers to the first word in the two word floating decimal value of the DIRECTORY entry. A "1" refers to the second word (the ten's exponent) of the two word floating decimal value of the DIRECTORY entry. The "W103" in line M010 refers to the ten's exponent of W03 in the DIRECTORY. The "W003" in line M011 refers to the significant digits of W03 in the DIRECTORY. Similarly, if the 9th digit is W, the 11th and 12th digits contain the DIRECTORY reference, and the 10th digit must be "0" or "1."

c) Any other letter in the 3rd or 9th digit positions must be the same as a storage area that has been specified somewhere in the pseudo-code. The two B's in line M009 of the example refer to the storage area reserved by the COMPUTER sentence.

6. Any transfers of control to within the COMPUTER section are made with M references, as in line M006.

Any transfers of control to other pseudo-code sentences call for the use of XXCN, as in lines M007 and M012. Wherever an XXCN appears in a COMPUTER section before the "ENDΔOWNΔΔΔΔΔ" sentinel, the left half of the word containing the XXCN must be a skip instruction, and the entire word following must be skip instructions.

Each XXCN referred to in the COMPUTER section before the "ENDΔOWNΔΔΔΔΔ" sentinel must appear at the beginning of a word placed between the ENDΔOWNΔΔΔΔΔ and the ENDΔCOMPUTER sentinels. In the 8th through the 10th digit positions of these words appears the number of the sentence to which control is to be transferred. If the number of the sentence to which control is transferred has an appended alphabetic, the appended alphabetic appears in the 11th digit position.

8. If the user wishes to maintain a UNIVAC word undisturbed by segmentation he should use an "XXST" address. Lines M006 and M014 in the example refer to "00ST." If more than one "XXST" is desired in one COMPUTER section, the "XXST's" must be numbered in order; 00ST, 01ST, 02ST, etc. If any "XXST's" are used, the word "STORAGE000nn" must appear immediately following the sentinel "ENDΔOWNΔΔΔΔΔ." The "nn" in "STORAGE000nn" is the number of XXST's in the COMPUTER section.

9. If the user desires to keep several consecutive lines anywhere in his COMPUTER section unmodified throughout compilation, he should insert the word "CONSTANTS XXX" just before the unmodified lines. The word "CONSTANTS XXX" does not take an "M" reference and will not appear on the compiled running tape. "XXX" is the number of lines following that are not to be modified.

VI

OPERATING INSTRUCTIONS AND SERVICE ROUTINES

I OPERATING INSTRUCTIONS

To perform a MATH-MATIC compilation mount tapes on the following servos:

SERVO	TAPE
1	MATH-MATIC MASTER
2	BLANK
3	BLANK
4	BLANK
5	BLANK
6	MATH-MATIC LIBRARY
7	BLANK
8	PROBLEM STATEMENT IN MATH-MATIC PSEUDO-CODE
9	BLANK

No breakpoints

Block subdivide Servo 3

Set Supervisory Control Printer on Normal

Initial read Servo 1

The following normal printouts will take place;

```
END PHASE 1

END PHASE 2
                               ⎧ one for
                               ⎪ each in-
SENTENCE #nnnn SERVO t         ⎨ put-out-
                               ⎪ put sen-
... ... ... ... ...            ⎩ tence

... ... ... ... ...

END PHASE 3

END PHASE 4

END SWEEP 1

END SWEEP 2

END SWEEP 3

END SWEEP 4

XXX BLOCKS OF RUNNING TAPE ON
SERVO 7.

BLOCK SUBDIVIDE SERVO 3 FOR
EDITED RECORD

XXX BLOCKS OF EDITED RECORD
ON SERVO 3.

END MATH-MATIC COMPILATION
```

The edited record should be removed from Servo 3 for printing. The compiled running tape may be left on Servo 7 or it may be mounted on any other servo for the problem run.

To perform the problem run, the user should mount his prepared input tapes and blanks for his output tapes according to the compilation printouts. Output servos should be block subdivided if high speed printing is intended. The servo selector key for the servo on which the running tape is mounted should be depressed. This servo is then initial read to begin the problem run. The only normal printouts during the problem run are those requested by the pseudo-code. At the completion of the problem run, the following printouts will take place:

```
    000000000XXX

    BLOCKS OF OUTPUT SERVO T

    ... ... ... ... ... ...

    ... ... ... ... ... ...

    END OF RUN.
```
(One set for each completed output.)

Each output tape is backward read and checked for readability, and all tapes are rewound.

All error printouts and correction options during compilation and during the problem run are discussed in the appendix of this manual.

II THE EDITED RECORD

The Edited Record provides a complete record of the transition from MATH-MATIC pseudo-code sentences through ARITH-MATIC pseudo-code operations into the final Univac C-10 running program. This Record will aid the user in detecting logical flaws in his program, and enable him to trace them back to the original pseudo-code sentences.

Upon completion of compilation the following will be printed out on the Supervisory Control typewriter:

```
XXX BLOCKS OF EDITED RECORD
ON SERVO 3.
```

(XXX specifies the number of blocks on the tape). The tape on servo 3 should then be removed and printed on the High Speed Printer.

The general format of the record is as follows:

1. The following header identifies the print-out itself:

   MATH-MATIC RECORD

2. The second header indicates the MATH-MATIC input:

   MATH-MATIC PSEUDO CODE

Under this header, the MATH-MATIC pseudo-code appears as written by the user.

3. The third header is:

INPUT AND OUTPUT STORAGE
AND SERVO ALLOCATION

Under this designation is given input and output sentences of the system as well as the variables contained in them as listed for storage purposes. This part of the record is divided into two sections.

A. The first section contains a list of all the input and output information broken down into five columns.

Column 1: Specifies the sentence number of the input or output pseudo-code sentence.

Column 2: The header of this column is SENT. NAME. The first word of every input or output sentence is listed in this column.

Column 3: This is headed SYMBOLIC ADDRESS and given the symbolic name of the area in memory in which the input or output data will be stored.

Column 4: This column is headed ACTUAL ADDRESS and under it is the actual address in Univac memory in which the data will be stored during the Univac run.

Column 5: The header of this column is SERVO ALLOCATION In this column is listed the servo number from which data will be read or to which data will be written during the Univac run.

B. The second section of INPUT AND OUTPUT STORAGE AND SERVO ALLOCATION will contain the storage address for every variable or constant used in the problem. These will be listed in three columns.

Column 1: This is headed PARAMETER. In this column is listed the name of each variable or constant.

Column 2: The header of this column is SYMBOLIC ADDRESS. Under it, the symbolic line number of the variable will be listed.

Column 3: ACTUAL ADDRESS is the header of the third column. This is a list of the line assigned to the parameter in the memory of Univac during the problem run.

4. The fourth of the main headers is "PERMANENT ARITH-MATIC CONTROLS."

Under this header is listed the Univac code which is turned out by the ARITH-MATIC compiler for every problem. This coding is used to control the problem during the problem run. The four different parts of the control section are labeled:

A. MEMORY CLEAR.

B. SPECIAL READS FOR DATA

C. SEGMENT CONTROL

D. FLOATING DECIMAL ARITHMETIC

E. CONSTANTS

In the next section of the record, the MATH-MATIC sentence is listed with its corresponding ARITH-MATIC and Univac code.

Each sentence will begin a new half page of High Speed Printer paper. At the top of the page the MATH-MATIC sentence number and the first word of the sentence will appear. Following the sentence number and name are two columns with the following headers:

ARITH-MATIC OPERATIONS

UNIVAC CODE

In column one the ARITH-MATIC operations which correspond to the MATH-MATIC sentence are listed and numbered.

In column two the Univac instructions which correspond to the MATH-MATIC sentence will be listed with their line numbers.

On the right side of the page substitution information will be listed only when two similar ARITH-MATIC operations appear in the same segment. The format of this information is:

SUBSTITUTION FOR XXX

BEGINS YYY

ENDS ZZZ

XXX is the call-word of the ARITH-MATIC operation.

YYY is the line number of the Univac coding where the operation begins.

ZZZ is the line number of the Univac where the operation ends.

The only coding that is listed in column 2 for a substituted operation will be coding to transfer the parameters to the proper storage locations and effect a transfer to the actual routine. This routine to perform the desired function is located at the end of the segment.

The ARITH-MATIC operations are numbered consecutively from zero to 99999. Univac instructions are numbered consecutively up to 999. The Univac addresses begin after the last address assigned to the storage blocks. When the number of instructions exceed 999 a new segment begins. Each segment is labeled. SEGMENT XXX appears under the header UNIVAC CODE and the line numbering will begin anew.

When a MATH-MATIC sentence involves an iterative loop or specifies a range of sentences for any other reason, part of the coding produced for that sentence is of necessity inserted at the end of the specified range, rather than in normal sequence. When this occurs, the word INSERT identifies the disassociated section of coding at the end of the specified range and a description of the nature of the ARITH-MATIC operations and UNIVAC instructions associated with the word INSERT appears next to it. Under this heading, the usual three-column listings of the ARITH-MATIC pseudo-code and Univac code is presented.

III SERVICE ROUTINES

Service routines associated with the MATH-MATIC system are described briefly in this section. Details and operating instructions for these service routines will appear in the MATH-MATIC PROGRAMMER'S MANUAL. The PROGRAMMER'S MANUAL will also contain the rules and conventions of writing glossaries and subroutines for the MATH-MATIC Library, and a detailed description of the Master and Library tapes and the present library glossaries and subroutines. With the information in the PROGRAMMER'S MANUAL the user can take advantage of the service routines to get more efficient use of the system.

a. Librarian

The Librarian routine is part of the MATH-MATIC Library tape. It permits the user to make changes, additions, and deletions in the glossaries and subroutines of the MATH-MATIC Library. The user must write the new glossaries and subroutines according to the rules of the MATH-MATIC system. The Librarian routine automatically inserts the new glossaries and subroutines into the Library at the appropriate positions. With the Librarian the user may expand the MATH-MATIC and ARITH-MATIC repertoires to facilitate handling of his special problems. The Librarian routine produces a complete new Library tape ready for use in compilation.

b. Pseudo-code Edit

The Pseudo-code Edit routine edits the problem statement in MATH-MATIC pseudo-code into a form which, when printed on the high speed printer, may be read and understood easily. Superscript symbols are translated into the actual numerical exponents and the pseudo-code sentences are lined up on the printed page. The Pseudo-code Edit routine will simplify the task of checking the problem statement on the tape. This routine is available separately.

c. Error Bypass

The MATH-MATIC system is equipped with the ability to detect various types of errors that might appear in a pseudo-code problem statement. In most cases these errors can be corrected during compilation by supervisory control type-ins. There are, however, certain errors which cannot be corrected conveniently during compilation, but necessitate correction of the original problem statement. This requires that compilation be interrupted, and restarted from the beginning after the problem statement has been corrected.

Phases I and II of the system contain options which permit the user to bypass errors that require rewriting the problem, and continue compilation through that phase. The user must, however, stop at the end of that phase and correct the pseudo-code. These error bypass options save the user from making several attempts to compile in order to locate all of his pseudo-code errors.

d. Locator

The locator routine will appear at the beginning of the MATH-MATIC master tape. It will enable the user to interrupt and to restart compilation at any phase or sweep. With instructions typed in by the user on the supervisory control, the locator will collect all of the intermediate output lists on one tape when compilation is interrupted, and redistribute these lists to the appropriate servos at the restart of compilation.

e. Automonitor

Occasionally the existence of a logical error in the pseudo-code problem statement is revealed only by the incorrect output produced by the problem run. In order to locate the error, the user will want to see the results of intermediate calculations during the problem run. The Automonitor routine on the MATH-MATIC Master tape will permit the user to compile a special running tape in which specified ranges of coding will be monitored during the problem run. The user can select which ranges he wishes monitored and whether the intermediate results should come out on tape or on the supervisory control.

APPENDIX

I DEFINITIONS

The following is a set of definitions of terms used frequently in the text.

1. ARGUMENT - A quantity or symbol of a quantity which is the subject of a mathematical or logical operation. An Argument is also called an operand.

   Examples;

SIN ALPHA	ALPHA is an argument
IF A = 10	A and 10 are arguments

2. BINARY OPERATION - An operation symbol which operates on two arguments.

   Examples:

A + B	+ is a binary operation
A - B	- is a binary operation
A * B	* is a binary operation

3. CHARACTER - Any of the 63 symbols recognized by UNIVAC. Those include the letters of the alphabet, the cardinal numbers, punctuation marks, and some other symbols.

4. CONSTANT - Any integer, decimal number, or number in a power of ten form containing 11 or less numeric characters.

5. CONTROL WORD - The first word in any input-output or control sentence.

6. EXPRESSION - A group of constants, variables and operations having mathematical meaning. Expressions are usually enclosed in parentheses.

7. FUNCTIONAL CALL-WORD - A symbol of a mathematical function appearing in an equation sentence.

   Examples:

COS BETA	COS is the functional call word for the function cosine.

8. OPERAND - See ARGUMENT

9. SENTENCE - A collection of words and symbols expressing a command which MATH-MATIC can execute. There are three types of sentences:

a. EQUATION - A collection of constants, variables, operations, and functional call words which define a new value of a variable in the MATH-MATIC SYSTEM.

b. CONTROL - A group of words, operands and symbols which determine the path through the MATH-MATIC pseudo-code.

c. INPUT-OUTPUT - A collection of words and variables interpreted by MATH-MATIC as a command involving tapes and/or data.

10. SUBSCRIPT - An integer or variable, enclosed by parentheses or commas, used to index another variable.

Examples:

X(I, 5)   I and 5 are subscripts

11. SUBSCRIPTED VARIABLE - Any variable with one, two, or three subscripts attached to it.

A(6, J) is a subscripted variable.

II REPERTOIRE OF ARITH-MATIC OPERATIONS

MATHEMATICAL OPERATIONS	DESCRIPTION	
AA0(A)(B)(C)	$A+B \longrightarrow C$	
AS0(A)(B)(C)	$A-B \longrightarrow C$	
AM0(A)(B)(C)	$A*B \longrightarrow C$	
AD0(A)(B)(C)	$A \div B \longrightarrow C$	
TS0(A)000(B)	$\sin A \longrightarrow B$	A in radians
TC0(A)000(B)	$\sin A \longrightarrow B$	A in radians
TT0(A)000(B)	$\tan A \longrightarrow B$	A in radians
TAT(A)000(B)	$\arctan A \longrightarrow B$	B in radians
HS0(A)000(B)	$\sinh A \longrightarrow B$	A in radians
HC0(A)000(B)	$\cosh A \longrightarrow B$	A in radians
HT0(A)000(B)	$\tanh A \longrightarrow B$	A in radians
ANI(A)000(B)	$-A \longrightarrow B$	
EXP(A)000(B)	$e^A \longrightarrow B$	
APN(A)(N)(B)	$A^N \longrightarrow B$;	N integral and N < 100
GPN(A)nnn(B)	$A^n \longrightarrow B$;	n positive and integral
X+A(N)($\log_{10}$A)(B)	$A^N \longrightarrow B$	
SQR(A)000(B)	$\sqrt{A} \longrightarrow B$	
RNA(A)(N)(B)	$\sqrt[N]{A} \longrightarrow B$;	N integral and < 100
GRN(A)nnn(B)	$\sqrt[n]{A} \longrightarrow B$;	nnn integral
LAU(A)($\log_{10}$B)(C)	$\log_B A \longrightarrow C$;	A>0

In all the above operations, (A) means the relative address of A.

CONTROL OPERATIONS	DESCRIPTION
AAL(X_i)(ΔX)(L_x) 01CN00(K)ΔΔ	$X_i + \Delta X \rightarrow X_i$ If $X_i < L_x$, transfer control to Opn. K If $X_i \geq L_x$ transfer control to next Opn.
ALL(X_i)(ΔX)(L_x) 01CN00(K)ΔΔ 02CN00(N)ΔΔ	$X_i + \Delta X \rightarrow X_i$ If $X_i \geq Lx$ transfer control to Opn. K If $X_i < Lx$ transfer control to Opn. N
ATL(X_i)(ΔX)(L_x) 01CN00(K)ΔΔ	$X_i + \Delta X \rightarrow X_i$ If $X_i \leq L_x$ transfer control to Opn. K If $X_i > L_x$ transfer control to next Opn.
AGL(X_i)(ΔX)(L_x) 01CN00(K)ΔΔ	$X_i + \Delta X \rightarrow c$ If $X_i \geq L_x$ and $c > L_x$, or if $X_i \geq L_x$ and $c < L_x$, $c \rightarrow X_i$ transfer control to next Opn.; otherwise $c \rightarrow X_i$ and transfer control to Opn. K.
QU0(A)(B)000 01CN00(K)ΔΔ	If A = B transfer control to Opn. K If A ≠ B transfer control to the next operation.
QUA(A)(B)000 01CN00(K)ΔΔ	If the $\|A\| = \|B\|$ transfer control to Opn. K If the $\|A\| = \|B\|$ transfer control to the next operation.
QT0(A)(B)000 01CN00(K)ΔΔ	If A > B transfer control to Opn. K If A ≤ B transfer control to the next operation.
QTA(A)(B)000 01CN00(K)ΔΔ	If $\|A\| > \|B\|$ transfer control to Opn. K If $\|A\| \leq \|B\|$ transfer control to the next operation.
0 QZ0AAAt(W)0000 R 01CN00(K)ΔΔ 02CN00(N)ΔΔ	AAA is the address of the input, t is the servo of the input, (K) is the Opn. number of the input GMI, W is used to rewind servo t, R is used to reset servo t, Control is transferred to Opn. N if Z... Z's are detected on the input tape; otherwise, control passes to the next operation.
U00000000000 01CN00(K)ΔΔ	Transfer control to Opn. K
JTC000000000 01CN00(K)ΔΔ 02CN00(N)ΔΔ	Control is transferred to Opn. K upon entering a JTC. Computation then proceeds from Opn. K to Opn. N.

CONTROL OPERATIONS	DESCRIPTION
03CN00(P)ΔΔ R00000000000	Operation N must be a special pseudo-operation whose call word is ROO. The ROO is set to transfer control to Opn. P only upon entering the JTC; if the ROO operation is executed other than going through a JTC, it will act as a skip. If set by a JTC, the operating containing the ROO is reset to a skip upon transferring control to Opn. P.

INPUT-OUTPUT OPERATIONS	DESCRIPTION
GTH($\substack{R\\W\\S}$)t($\substack{B\\F}$)NNNMMM	(R) read, (W) write, or (S) skip NNN blocks (B) backward or (F) forward on servo t, starting address MMM.
GMIBBtSSSAAA	Read from servo t into storage area BB SSS words and place the current SSS word item in AAA.
GMNO($\substack{H\\L}$)tSSSAAA	Write the array of item size SSS beginning with address AAA on servo t at (H) high density or (L) low density.
GMO0($\substack{H\\L}$)tSSSAAA	Write SSS words starting with address AAA on servo t at (H) high or (L) low density.
GMSBBtXXXAAA	Pick up XXX words beginning with address AAA edit them and place them in an output block beginning with address BB. When the output block is filled, write it on servo t.
GMTBBtXXXAAA	Pick up XXX words beginning with address AAA. Convert them and place them in output block beginning with address BB. When the output block is filled, write it on servo t.
GMUBBtXXXAAA	Pick up XXX words beginning with address AAA edit them for uniprinter and place them in output block beginning with address BB. When

	the output block is filled, write it on servo t.
GASBBtXXXAAA	Pick up the array of item size XXX beginning with address AAA. Edit it and place it in output area beginning with address BB. Write the array from output area to servo t.
GATBBtXXXAAA	Pick up the array of item size XXX beginning with address AAA. Convert it and place it in output area beginning with address BB. Write the array from the output area on servo t.
FILBBtSSS0$\left(\begin{smallmatrix}R\\0\end{smallmatrix}\right)$Z 01CN00(K)ΔΔ	Z = 1; high speed printer unedited. Z = 2; high speed printer edited. Z = 3; uniprinter unedited. Z = 4; uniprinter edited. BB is the output area, t is the servo, and SSS is the output item size. The remainder of the output block or an entire output item, whichever is larger, is filled with sentinels and written on servo t. The sentinels are Z... Z's for unedited output and 's for edited output. The number of blocks on the output tape is printed out and the output tape is backward read, checked, and rewound. Operation K is the write operation.
CLO0$\left(\begin{smallmatrix}0\\W\end{smallmatrix}\right)$t000000 01CN00(K)ΔΔ	The GMI in Opn. K is reset to start reading from the beginning of the block. If W is used, servo t is rewound.

SPECIAL OPERATIONS	DESCRIPTION
GMMAAA0SSBBB	Pick up SS words beginning with memory address AAA and place them in an area beginning with address BBB.
GTOAAA$\left(\begin{smallmatrix}BBB\\000\end{smallmatrix}\right)\left(\begin{smallmatrix}CCC\\000\end{smallmatrix}\right)$	Print out in two word floating decimal form one, two, or three values with addresses AAA, BBB, CCC.
GTIAAA$\left(\begin{smallmatrix}BBB\\000\end{smallmatrix}\right)\left(\begin{smallmatrix}CCC\\000\end{smallmatrix}\right)$	Type in two word floating decimal form into addresses AAA, BBB, or CCC one, two, or three values.

GWSttttt. . . 00	Rewind servos ttt. . . and stop.
SBIAAAIIISSS	Pick up the AAA-1 element of the subscripted array that begins with address III and place it in storage address SSS.
SBOAAAIIISSS	Pick up from SSS the AAA-1 element and place it in the subscripted array beginning with address III.

NON-OPERATIONAL SENTINELS	DESCRIPTION
XXXBLKΔSTGΔΔ A00000000040 B00000000050 F00000000120	XXX=number of storage areas listed. A, B, etc. are the storage areas. The last three digits give the size in number of words of the area.
PREΔSTGΔONΔt	t is the servo containing pre-read input
B0000000000	Storage area for pre-read
WKGΔSTORAGEΔ	
F00000000120	Storage area for working storage data
ENDΔSTORAGEΔ	No further storage listings; next line is operation #0.
BEGINLOOPXXX	Indicates the beginning of an iterative loop and is to precede the first operation constituting the loop. XXX represents the loop number.
ENDΔLOOPΔXXX	Indicates the end of an iterative loop and is inserted following the last operation of the loop. XXX is the loop number.
ENDΔCODINGΔΔ	Indicates the end of ARITH-MATIC pseudo-code.

III REPERTOIRE OF MATH-MATIC SENTENCES

The following is list of sample MATH-MATIC pseudo-code sentences illustrating most of the combinations of options available in the input-output and control sentences. The rules in chapter II and III are recommended whenever a question arises in writing pseudo-code.

1. READ A B C .
2. READ A B C IF SENTINEL JUMP TO SENTENCE 8 .
3. READ A B C D E IF SENTINEL RESET AND JUMP TO SENTENCE 10 .
4. READ C D E F IF SENTINEL REWIND AND JUMP TO SENTENCE 24 .
5. READ X Y SERVO 4 .
6. READ X Y Z A B F D STORAGE A .
7. READ A X B Y SERVO 1 STORAGE Q .
8. READ A B C STORAGE Z SERVO - IF SENTINEL RESET AND JUMP TO SENTENCE 73E .
9. READ-ARRAY X(6, 6) .
10. READ-ARRAY X(6, 8) SERVO 2 STORAGE C .
11. READ-ITEM X(8, 10) .
12. READ-ITEM A(5, 5, 5) IF SENTINEL JUMP TO SENTENCE 10A .
13. READ-ITEM BETA(40) SERVO 8 STORAGE B IF SENTINEL REWIND AND JUMP TO SENTENCE 3 .
14. WRITE A B C .
15. WRITE FOR UNIPRINTER A B C .
16. WRITE EDIT A B C .
17. WRITE CONVERT A B C .
18. WRITE EDIT FOR UNIPRINTER A B C .
19. WRITE A B C SERVO 6 .
20. WRITE FOR UNIPRINTER A X B Y STORAGE F .
21. WRITE AND EDIT X Y Z SERVO 4 STORAGE Y .
22. WRITE CONVERTED A B D G F L N STORAGE K SERVO 3 .
23. WRITE-ARRAY X(20) .
24. WRITE-ARRAY FOR UNIPRINTER MATRIX(5, 10) .
25. WRITE-ARRAY EDIT Z(5, 5, 3) .
26. WRITE-ARRAY CONVERT AB(7, 4) .
27. WRITE-ARRAY EDIT FOR UNIPRINTER C(9) .
28. WRITE-ARRAY EDIT AND CONTAIN X(5)
29. WRITE-ARRAY CONVERT AND CONTAIN A(10, 10) .
30. WRITE-ARRAY EDIT FOR UNIPRINTER AND CONTAIN B(25).
31. WRITE-ARRAY A(6, 6, 6) SERVO 1 .
32. WRITE-ARRAY EDITED Z(5, 5) STORAGE A .
33. WRITE-ARRAY AND CONVERT Z(4, 2, 2) SERVO 6 .
34. WRITE-ARRAY EDIT FOR UNIPRINTER AND CONTAIN X(8, 8) SERVO 3 STORAGE N .
35. WRITE-ITEM X(10, 10) .
36. WRITE-ITEM FOR UNIPRINTER X(5, 5) .
37. WRITE-ITEM EDIT A(110) .
38. WRITE-ITEM CONVERT B(65) .
39. WRITE-ITEM EDIT FOR UNIPRINTER C(50, 3) .
40. WRITE-ITEM X(10, 10) SERVO 5 .
41. WRITE-ITEM FOR UNIPRINTER X(10, 5) SERVO 7 .
42. WRITE-ITEM EDITED C(40) STORAGE L .
43. WRITE-ITEM AND CONVERT F(4, 4, 4) SERVO 4 STORAGE D .
44. WRITE-ITEM EDIT FOR UNIPRINTER X(6, 6) STORAGE P SERVO 6 .

45. CLOSE-INPUT SENTENCE 10 .
46. CLOSE-INPUT AND REWIND SENTENCE 1 .
47. CLOSE-OUTPUT SENTENCE 24D .
48. CLOSE-OUTPUT AND REWIND SENTENCE 6 .

Control sentences

1. CONTAIN X(25) .
2. CONTAIN A(7, 7) STORAGE A .
3. EXECUTE SENTENCE 24 .
4. EXECUTE SENTENCE 3 TO 10 .
5. IF A < 17.3, JUMP TO SENTENCE 17 .
6. IF A = B, JUMP TO SENTENCE 12A .
7. IF A > 10, JUMP TO SENTENCE 14 .
8. IF A < B JUMP TO SENTENCE 3, IF A = B, JUMP TO SENTENCE 6, IF A > B JUMP TO SENTENCE 35 .
9. IF ALPHA = BETA JUMP TO SENTENCE 15 IF ALPHA > BETA JUMP TO SENTENCE 29 .
10. IF|A|<|B|JUMP TO SENTENCE 5.
11. IF|A|=|B|JUMP TO SENTENCE 6.
12. IF|B|>|A|JUMP TO SENTENCE 29.
13. IF|ALPHA|=|BETA|JUMP TO SENTENCE 6 IF|ALPHA|> |BETA|JUMP TO SENTENCE 24.
14. IF A(I) < B JUMP TO SENTENCE 25 .
15. IF A(5, 4) = 23 JUMP TO SENTENCE 7 .
16. IF C > X(4, 3, J) JUMP TO SENTENCE 5 .
17. IF A(I) < B(J) JUMP TO SENTENCE 34 IF A(I) = B(J) JUMP TO SENTENCE 51 IF A(I) > B(J) JUMP TO SENTENCE 68 .
18. IF A = > B JUMP TO SENTENCE 10 .
19. IF A(I) = < B(J) JUMP TO SENTENCE 10 .
20. JUMP TO SENTENCE 3G .
21. PRINT-OUT A B X Y .
22. TYPE-IN ALPHA, BETA, K .
23. VARY X 10 (0.05) 25 SENTENCE 7 TO 15 .
24. VARY X 10 (D) 25 Y 0 (L) ALPHA SENTENCE 20 THRU 21 .
25. VARY I 1 (1) 10 J 11 (1) 20 K 1 (1) LASTVALUE SENTENCE 10 THRU 15 .
26. VARY X 1, 3, 6, 10, 11.5, 15 SENTENCE 6 TO 19E .
27. VARY X Y Z 1 2 3 2 4 7 5 9 26 SENTENCE 1 TO 25 .
28. STOP .
29. IGNORE .

IV MODIFIED UNITYPER KEYBOARD

With a modified UNITYPER keyboard, the digit or symbol in the first column may be unityped directly on the tape, and the typed digit or symbol appears on the hard copy. The modified UNITYPER keyboard may be obtained from Remington Rand Univac at nominal cost. These digits and symbols do not appear on the ordinary UNITYPER Keyboard. The character in the second column must be typed in this case. MATH-MATIC interprets the characters as the digit or symbol in the first column.

INTENDED DIGIT OR SYMBOL ON MODIFIED KEYBOARD		CHARACTER ON ORDINARY KEYBOARD
superscript	0	Σ
superscript	1	β
superscript	2	t̸
superscript	3	+̸
superscript	4	!'
superscript	5	#
superscript	6	S
superscript	7	%
superscript	8	¢
superscript	9	?
superscript	-	i̸
superscript	.	:
superscript	/	'
> (greater than)		&
< (less than)		@

V MATH-MATIC PRINT-OUTS

This section contains both the normal and error print-outs which may occur during compilation and running of the program tape. Each print-out contains a carriage return in the first digit position of the first word so that the Supervisory Control Printer may be operated on normal. The print-outs are listed according to Phase, Glossary, Sweep, A-3 Routines, System Restrictions, and Running Program Tape Errors. The exact form of the print-out, except for the carriage return, is listed in the first column. The second column contains a description of the error that caused the print-out, and the third column the procedure that must be taken in order to correct the error.

The print-outs of Phase 1 are cataloged and will always begin with the word PRT-OUT XX followed by SENT. NO. nnnn, where XX is the number of the print-out and nnnn the number of the sentence where the error occurred. A breakpoint option permits the user to by-pass those errors that require a re-writing of the pseudo code. Selecting this option allows the user to error check the remaining sentences of his problem, but does not allow him to go on to Phase 2.

The print-outs of Phase 2 are listed as main body and Glossary. The print-outs cataloged as main body are those which are common to all types of sentences. Glossary print-outs are those which are common to a particular sentence type. The first word of a Glossary print-out will contain the name of the Glossary in which the error occured and the second word the number of the sentence. A breakpoint option is provided for those print-outs that require a rewriting of the pseudo code. In selecting this option the user may not go on to Phase 3 before correcting his errors.

Since there are few error print-outs in Phase 3, Phase 4, and Sweep 1, the print-outs are listed only by Phase and Sweeps. Sweeps 2, 3, and 4 have no error print-outs. A-3 Routine print-outs only occur from compiler sections. Running program errors arise from the improper manipulation of data, that is, calculations that lead to division by zero, taking the arcsine of a quantity greater than one, etc.

The user is strongly advised to carefully analyze his error before taking steps to correct it and to examine the pseudo code to ascertain whether or not other portions of the problem will be affected by any changes made.

NORMAL PRINT-OUTS

PRINT-OUT	DESCRIPTION	PROCEDURE
1. END PHASE 1	Phase 1 of compilation has been completed.	NONE
2. NO DIRECTORY LISTED	A computer or compiler sentence is listed in the pseudo coding but the COMP sections do not have an accompanying DIRECTORY.	NONE
3. END PHASE 2	Phase 2 of compilation has been completed.	NONE
4. SENT. NO. nnnn SERVO t	The prepared input or output data required by sentence number nnnn will be either mounted or written on servo t.	Mount data on servo t if input is required, or mount a blank on servo t if output is being written on servo t during the problem run.
5. REMOVE SERVO 4 MOUNT BLANK	During compilation data, which is needed for the running program tape, is now on servo 4.	Remove the tape on Servo 4 and save it for the problem run, and mount a blank tape to continue compilation.
6. END PHASE 3	Phase 3 of compilation has been completed.	NONE
7. END SWEEP 1	SWEEP 1 of compilation has been completed.	NONE
8. END SWEEP 2	Sweep 2 of compilation has been completed.	NONE

9. END SWEEP 3	Sweep 3 of compilation has been completed.	NONE
10. END SWEEP 4	Sweep 4 of compilation has been completed.	NONE
11. XX BLOCKS OF RUNNING TAPE ON SERVO 7	Servo 7 which is now the running program tape contains XX blocks of C-10 coding.	NONE
12. XX BLOCKS OF EDITED RECORD ON SERVO 3	Servo 3 contains XX blocks of the EDITED RECORD.	NONE
13. END MATH-MATIC COM-PILATION	Compilation has been completed.	NONE

PHASE 1 ERROR PRINT-OUTS

PRINT-OUT	DESCRIPTION	PROCEDURE
1. PRT-OUT 01 SENT. NO. nnnn NO PAREN	The first sentence in the pseudo code has no left paren; nnnn will be spaces and not the number of the first sentence.	Hit start bar to continue.
2. PRT-OUT 02 SENT. NO. nnnn NXT SENT NO. PAREN MSNG	The sentence number of the sentence following nnnn has no right paren after the sentence number.	Hit start bar to continue.
3. PRT-OUT 03 SENT. NO. nnnn NXT SENT. NO. PAREN BUT NO SPACE	The sentence number of the sentence following nnnn has no space after the right paren of the sentence number.	Hit start bar to continue.
4. PRT-OUT 04 SENT. NO. nnnn NXT SENT. NO. EQUAL SIGN SPACE MSNG	A space is missing either before or after the equal sign of sentence number nnnn. This print-out occurs only in sentences that are equations.	Hit start bar to continue.

PRINT-OUT	DESCRIPTION	PROCEDURE
or		
PRT-OUT 04 SENT. NO. nnnn EQUAL SIGN SPACE MSNG		
5. PRT-OUT 05 SENT. NO. nnnn SPACE PERID NO PAREN (XXXXXXXXXXXX) (YYYYYYYYYYYY) where X. . . X and Y. . . Y is that portion of the pseudo-code where the error occurred.	A space period (Δ.) signifying the end of the pseudo-code sentence is found by MATH-MATIC, but a left paren indicating the beginning of a new sentence has not been found. Therefore, either the left paren of the sentence number following nnnn has been omitted, or a decimal point exists within SENT. NO. nnnn.	
	If the left paren is missing. . . .	Hit start bar to begin processing the sentence after SENT. NO. nnnn.
	If a decimal point exists within sentence nnnn, that is, between the sentence number and the space period which indicates the end of the sentence. . .	Set breakpoint 8 and force transfer. MATH-MATIC will continue processing SENT. NO. nnnn.
6. PRT-OUT 06 SENT. NO. nnnn NXT SENT. NO. (0000000nnnn) or (nnnnΔΔΔΔΔΔΔΔ) INCORRECT TYP-IN CORR	The sentence number following sentence nnnn is either less than or equal to nnnn; or it contains an alphabetic as its first digit; or it is greater that 999Z.	Type in: X. . . . XΔΔΔΔΔΔΔΔΔΔ where X is the correct form of the sentence number.
7. PRT-OUT 07 SENT. NO. nnnn SENT. NAME (X. . . XΔΔΔΔΔΔΔ) INCORRECT TYP-IN CORR NEXT WORD	This print-out results only from errors associated with the first word of a statement. The word X. . . XΔΔΔΔΔΔΔ was not found in the MATH-MATIC catalog, and, therefore, must be incorrectly spelled or a word which is not yet part of the repertoire.	

PRINT-OUT	DESCRIPTION	PROCEDURE
	If the print-out resulted from the latter, the problem must be re-written within the present repertoire.	
	If the sentence name is incorrectly spelled, the corrections are to be typed in a word at a time in answer to the print-out "NEXT WORD." Upon typing in the last correction and in answer to "NEXT WORD," a word of spaces must be typed in to terminate the correction routine.	
	If, for example, EXECTEΔΔΔΔΔΔ is printed out....	Type in: EXECUTEΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ
	If, however, READXΔΔΔΔΔΔΔ is printed out and resulted from ommitting the space between READ and X...	Type in: READΔΔΔΔΔΔΔΔ XΔΔΔΔΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ
8. PRT-OUT 08 SENT. NO. nnnn PSEUDO WORD (X...XΔΔΔΔΔΔΔ) INCORRECT TYP-IN CORR NEXT WORD	This print-out results only from errors associated with functional call words. The word printed out was interpreted as a functional call word and was not found in the catalog. To continue the problem run, the correct form of the call word and in some cases its associated arguments must be typed in. It is of utmost importance to check the original pseudo coding to determine the exact cause of the error. Corrections are to be typed in a word at a time in answer to "NEXT WORD". Upon making the last correction and in answer to "NEXT WORD", type in a word of spaces to terminate the correction routine.	

PRINT-OUT	DESCRIPTION	PROCEDURE

Examples of common errors associated with funcitonal call words.

PSEUDO WORD PRINTED OUT	CORRECT FORM OF THE PSEUDO WORD	
(1) CSSΔΔΔΔΔΔΔΔΔ A mispelled functional call word.	(1) COSΔALPHA	(1) Type in COSΔΔΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ
(2) 0.123EXPΔΔΔΔ Operation sign missing between 0.123 and EXP	(2) 0.123+EXPΔ0.5	(2) Type in: 0.123ΔΔΔΔΔΔΔ +ΔΔΔΔΔΔΔΔΔΔΔ EXPΔΔΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ
(3) -0.779ΔΔΔΔΔ -0.7794 is the converted exponent and the error resulted from omitting the operation sign between the exponent and the functional call word.	(3) X-0.7794+ SINΔB	(3) Type in: -0.7794ΔΔΔΔΔ +ΔΔΔΔΔΔΔΔΔΔΔ SINΔΔΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ
(4) ALPHAROOTΔΔΔ Omitting a space between the first argument of a two argument functional call word.	(4) ALPHAΔROOTΔX	(4) Type in: ALPHAΔΔΔΔΔΔΔ ROOTΔΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ

NOTE:
The omission of a space between the second argument of a two argument call word cannot be corrected by type-ins, that is, if AΔΔΔΔΔΔΔΔΔΔΔ is printed out and the intended form

PRINT-OUT	DESCRIPTION	PROCEDURE
	is AΔPOWΔB, this would indicate a space was omitted between POW and B. To by-pass this error, SCICR to the error by-pass routine. Setting breakpoint 1 and forcing transfer will permit an error check of the remaining sentences. The memory location of the error by-pass routine is provided with the MATH-MATIC master tape.	
9. PRT-OUT 09 SENT. NO. nnnn TYPE-IN INCOR-RECT TYP-IN CORR AGAIN NEXT WORD	This print-out will result only from incorrectly typing in the sentence name or functional call word related to PRT-OUT 07 and PRT-OUT 08 above.	Check the pseudo-code and make certain the error has been properly analyzed, and then type-in again the correction associated with the error.
10. PRT-OUT 10 SENT. NO. nnnn DECI ALPHA (.X...XΔΔΔΔΔΔ) TYP-IN CORR	The pseudo word .X...XΔΔΔΔΔΔ has an alphabetic after the decimal point. All decimaled quantities must be numeric.	Type in: 0.X...XΔΔΔΔΔ
	This print-out will also result when a pseudo word appearing in a statement contains as its first digit a decimal point.	Type in: X...XΔΔΔΔΔΔΔ
11. PRT-OUT 11 SENT. NO. nnnn LFT NUMBER NUMERIC TYP-IN CORR	The left member of an equation is a numeric quantity, that is, it contains a numeric in its first digit position.	Type in: X...XΔΔΔΔΔΔΔ
12. PRT-OUT 12 SENT. NO. nnnn PAREN NOT PAIRED REWRITE	The parens of sentence nnnn not properly paired. To insure accurate results in the other phases, the pseudo-code must be re-written before continuing.	Hit start bar and MATH-MATIC will rewind all tapes or Set breakpoint 1;

PRINT-OUT	DESCRIPTION	PROCEDURE
		force transfer; and MATH-MATIC will begin processing the sentence following nnnn.
13. PRT-OUT 13 SENT. NO. nnnn ABS SIGN NOT PAIRED REWRITE	The absolute signs of sentence nnnn are not properly paired.	Hit start bar and MATH-MATIC will wind all tapes or Set breakpoint 1; force transfer; and MATH-MATIC will begin processing the sentence following nnnn.
14. PRT-OUT 14 SENT. NO. nnnn PSEUDO WORD (XXXXXXXXXXXX) (YYYYYYYYYYYY) TOO LONG REWRITE	MATH-MATIC has detected a pseudo word of more than 12 digits in sentence nnnn. A common error associated with this print-out is the omission of a space or operation sign between pseudo words. X. . . X and and Y. . . Y will give that portion of the pseudo code where the error occurred.	Hit start bar and MATH-MATIC will rewind all tapes. or Set breakpoint 1; force transfer; and MATH-MATIC will begin processing the sentence following nnnn.
15. PRT-OUT 15 SENT. NO. nnnn SUB VARBLE (X. . . XΔΔΔΔΔΔΔ) NUMERIC REWRITE	The subscripted variable of sentence nnnn is numeric instead of alphabetic.	Hit start bar and MATH-MATIC will rewind all tapes or Set breakpoint 1; force transfer; and MATH-MATIC will begin processing the sentence following nnnn.
16. PRT-OUT 16 SENT. NO. nnnn	The numerical exponent of sentence nnnn is not of the power of	Hit start bar and MATH-MATIC will

PRINT-OUT	DESCRIPTION	PROCEDURE
NUM. EXP. INCORRECT REWRITE	ten form. The only exponents permitted in statements are those in the power of ten notation.	rewind all tapes. or Set breakpoint 1; force transfer; and MATH-MATIC will begin processing the sentence after nnnn.
17. PRT-OUT 17 SENT. NO. nnnn REWRITE PSEUDO CODE END PHASE 01	This print-out will result only when an error requiring a re-writing of the pseudo-code has been encountered, and the programmer by using breakpoint 1 decided to error check the remaining pseudo-code. Make necessary corrections and begin a new MATH-MATIC compilation.	Hit start bar to re-wind all tapes.
18. WRONG TAPE ON SERVO -6 MOUNT AT-3 LIBRARY	The MATH-MATIC library tape is not mounted on Servo 6.	Hit start bar to re-wind all tapes. Mount the library tape and begin a new compilation.
19. PRT-OUT 19 DIRECTORY OR COMP SECT MISSING REWRITE	A computer and/or compiler sentence is listed in the pseudo code sentences, signifying that a directory and/or COMP section will follow the pseudo code. Neither the directory or the COMP section was found in the first block following the pseudo code sentences.	Hit start bar to re-wind tapes.
20. PRT-OUT 20 DIRECTORY TOO LONG REWRITE	The directory contains more than 99 entries, exclusive of header and sentinel. The directory may be shortened by listing some of the constants within a COMP section.	Hit start bar to re-wind tapes.
21. PRT-OUT 21 DIRECTORY WORD XXX NUM. EXP. INCORRECT	The only exponents permitted in the directory are those of the power of ten notation. If the error can be corrected by	Type in:

PRINT-OUT	DESCRIPTION	PROCEDURE
(X. . . X) TYP-IN CORR	typing in the correct power of ten format. . .	X. . . XΔΔΔΔΔΔΔ where X. . . X is the corrected word.
where X... X is incorrect word	If the error cannot be corrected by typing in a new word . . .	Type in: ΔΔΔΔΔΔΔΔΔΔΔΔ PRT-OUT 23 will occur giving the programmer a breakpoint option.
22. PRT-OUT 22 DIRECTORY REWRITE DIRECTORY	This print-out will occur only if the programmer elected to force transfer on breakpoint option in order to check the remainder of the directory.	Hit start bar to rewind tapes.
23. PRT-OUT 23 DIRECTORY BREAKPOINT OPTION	This print-out will result only if a word of spaces is typed in instead of a new word in PRT-OUT 21. If the breakpoint option is elected, PRT-OUT 22 will result after the directory is error checked.	Hit start bar to rewind tapes. or Set breakpoint 1; force transfer to continue error checking the DIRECTORY.

PHASE 2 ERROR PRINTOUTS

PRINT-OUT	DESCRIPTION	PROCEDURE
1. ARGMT MSNG SENT. NO. nnnn REWRITE (sΔΔΔΔΔΔΔΔΔΔΔ)	The argument before or after an operation symbol of the equation listed as sentence number nnnn is missing; s is the operation symbol that has a missing argument. NOTE: In selecting the breakpoint 2 option, set comma breakpoint to stop compilation at the end of Phase 2. The breakpoint option permits the user to error check his pseudo coding. The errors must be corrected before a new compilation may be attempted.	Hit start bar to rewind tapes. or Set breakpoint 2; force transfer to by-pass the error.

PRINT OUT	DESCRIPTION	PROCEDURE
2. WORD MSNG SENT. NO. nnnn REWRITE	The sentence listed as number nnn is incomplete in that the GLOSSARY required by the sentence expects more information than is contained in the statement.	Hit start bar to rewind tapes. or
	See NOTE of Print-Out 1 for breakpoint option.	Set breakpoint 2; force transfer to bypass the error.
3. PAREN MSNG SENT. NO. nnnn REWRITE	The parentheses of sentence number nnnn are improperly paired, that is, they are not closed.	Hit start bar to rewind tapes. or
	See NOTE of Print-Out 1 for breakpoint option.	Set breakpoint 2; force transfer to bypass the error.
4. OPERATION SYMBOL MSNG SENT. NO. nnnn REWRITE	An operation symbol is missing between two arguments of the equation listed as sentence number nnnn.	Hit start bar to rewind tapes. or
	See NOTE of Print-Out 1 for breakpoint option.	Set breakpoint 2; force transfer to bypass the error.
5. WORD AFTER STORAGE MISSING SENT. NO. nnnn TYP-IN CORR	The pseudo word STORAGE of sentence number nnnn is not followed by a storage area designation.	Type in: X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ where X is the alphabetic storage area designated.
6. WORD AFTER SERVO MISSING SENT. NO. nnnn TYP-IN CORR	The pseudo word SERVO of sentence number nnnn is not followed by a servo number.	Type in: X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ where X is the correct servo number.

	PRINT-OUT	DESCRIPTION	PROCEDURE
1.	CLOSE-INPUT or CLOSE-OUTPUT SENT. NO. nnnn SENT. REF. MISSING TYP-IN CORR	A CLOSE-OUTPUT sentence must make a reference to some output sentence. This print-out will result when the sentence number of an output sentence is omitted from the CLOSE-OUTPUT sentence.	Type in: X. . . X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ where X. . . X is the sentence number that is referred to by the CLOSE-OUT sentence.
2.	CLOSE-INPUT or CLOSE-OUTPUT SENT. NO.nnnn WORD AFTER AND MSNG TYP-IN CORR	The word AND is not followed by REWIND indicating either an unintentional omission of the word REWIND or possibly an incomplete deletion.	Type in: (1) REWIND$\underline{\Delta\Delta\Delta\Delta\Delta\Delta}$ if the word was omitted, or (2) $\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ if the REWIND option is not desired.
3.	CLOSE-INPUT or CLOSE-OUTPUT SENT. NO. nnnn FORMAT INCORRECT TYP-IN CORR	The sentence does not conform to the prescribed format.	Type in the entire sentence, omitting the name of the sentence, a pseudo-word per Univac word. X. . . X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ When finished type in a word of spaces.
4.	CONTAIN or READ-ARRAY SENT. NO.nnnn VARIABLE (X. . . X) NOT ALPHA TYP-IN CORR	The variable is not alphabetic.	Type in: X. . . X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ where X. . . X is the correct form of the variable.
5.	CONTAIN or	The variable of the subscripted array is missing.	Type in: X. . . X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$

PRINT-OUT	DESCRIPTION	PROCEDURE
READ-ARRAY SENT. NO. nnnn VARIABLE MISSING TYP-IN CORR		where X...X is the intended variable.
6. CONTAIN or READ-ARRAY SENT. NO. nnnn PAREN MSNG TYP-IN CORR	A paren is missing from the subscripted array	Type in: (1) (variable) ΔΔΔΔ (2) (first subscript)Δ (3) (second subscript)Δ (4) (third subscript)Δ If there are no second and/or third subscripts type in a word of spaces for the third and fourth words of the four word type-in.
7. CONTAIN or READ-ARRAY SENT. NO. nnnn ARRAY SIZE (X...X) ALPHABETIC TYP-IN CORR	The size of the array is X...X, and it is alphabetic. In a READ-ARRAY sentence the array size must be numeric.	Type in: X...XΔΔΔΔΔΔΔ where X...X is the correct array size.
8. CONTAIN or READ-ARRAY SENT. NO. nnnn MORE THAN 3 SUBSCRIPTS REWRITE	There are more than three subscripts in the array of sentence number nnnn. See NOTE of Print-Out 1 of Phase 2 for explanation of breakpoint option.	Hit start bar to rewind tapes. or Set breakpoint 2; force transfer to by pass the error.
9. CONTAIN or	There are more than 250 elements of the array in sentence number nnnn.	Hit start bar to rewind tapes. or

	PRINT-OUT	DESCRIPTION	PROCEDURE
	READ-ARRAY SENT. NO. nnnn ARRAY SIZE TOO LARGE REWRITE	See NOTE of Print-Out 1 of Phase 2 for an explanation of the breakpoint option.	Set breakpoint 2; force transfer to by pass the error.
10.	COMP LABEL OF SENT. NO. nnnn MISSING (COMPUTERΔΔΔΔ) or (COMPILERΔΔΔΔ) TYP-IN CORR	The label of COMPUTER or COMPILER is missing. This error usually comes about from omitting the dash in the label.	Type in: COMPUTER-XX or COMPILER-XXX where XXX is the correct label.
11.	COMP SENT. NO. nnnn ALPHABETIC STORAGE SIZE (X...X) TYP-IN CORR	The size of the storage area requested by the COMP sentence is an alphabetic character instead of a numeric.	Type in: XXXΔΔΔΔΔΔΔΔΔ where XXX is the correct storage size.
12.	COMP SENT. NO. nnnn PAREN OF STORG SIZE MISSING (X...X)	A paren is missing from the storage area size requested by sentence number nnnn.	
		If X...X is the intended storage size...	Set breakpoint 2; force transfer to continue.
		If X...X is not the intended storage size...	Hit start bar to rewind the tapes.
13.	COMP SENT. NO. nnnn STORG AREA (XΔΔΔΔΔΔΔΔΔΔΔ) NOT ALPHA TYP-IN CORR	The storage area symbol X is not an alphabetic character.	Type in: XΔΔΔΔΔΔΔΔΔΔΔ where X is an alphabetic storage area symbol.
14.	COMP SENT. NO. nnnn SERVO NO. (XΔΔΔΔΔΔΔΔΔΔΔ)	The servo being assigned by the COMP sentence is not 1 thru 9 or minus (-).	Type in: XΔΔΔΔΔΔΔΔΔΔΔ

	PRINT-OUT	DESCRIPTION	PROCEDURE
	INCORRECT TYP-IN CORR		where X is the proper servo number.
15.	EXECUTE SENT. NO. nnnn SENT REF MISSING TYP-IN CORR	The sentence range over which the sequence of execution is to be altered is missing in sentence number nnnn.	Type in: X...XΔΔΔΔΔΔΔ Y...YΔΔΔΔΔΔΔ where X...X and Y...Y is the sentence range. If the range is one sentence only, type in a word of spaces for Y...Y.
16.	IF OPTN xx SENT. NO.nnnn SIGN MSNG TYP-IN CORR	The relation sign is missing from clause xx of the IF statement listed as sentence number nnnn.	Type in: X...XΔΔΔΔΔΔΔ Y...YΔΔΔΔΔΔΔ Z...ZΔΔΔΔΔΔΔ where X...X and Z...Z are the intended variables or numbers and Y...Y is the intended relation sign or combination of relation signs.
17.	IF OPTN xx SENT. NO. nnnn FORMAT INCORRECT TYP-IN CORR	The xx clause of the IF statement listed as sentence number nnnn is not of the format prescribed in the manual.	Type in IFΔΔΔΔΔΔΔΔΔΔ if there is another "IF" clause in the sentence. Otherwise type in spaces.
18.	IF SENT. NO. nnnn SENT REF MISSING TYP-IN CORR	The sentence number to which control is to be transferred is missing.	Type in: X...XΔΔΔΔΔΔΔ where X...X is the number of the sentence to which control is to be transferred.
19.	POW SENT. NO. nnnn ARGMT MSNG REWRITE *ΔΔΔΔΔΔΔΔΔΔΔ	This print-out will result only where there is an argument missing from the multiplication sign (*) in a numeric written as a power of ten. See NOTE of Print-Out 1 of Phase 2 for an explanation of the breakpoint option.	Hit start bar to rewind tapes; or Set breakpoint 2; force transfer to bypass the error.

PRINT-OUT	DESCRIPTION	PROCEDURE
20. READ SENT. NO. nnnn TOO MANY VARIABLES REWRITE	There are more than 50 variables listed in the READ statement.	Hit start bar to rewind tapes;
		or
	See Note of Print-Out 1 of Phase 2 for an explanation of the breakpoint option.	Set breakpoint 2; force transfer to by-pass the error.
21. READ SENT. NO. nnnn SENT REF MISSING TYP-IN CORR	The READ statement listed as sentence number nnnn used the IF SENTINEL JUMP option but does not list the sentence number to which control is to be transferred.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the missing sentence number.
22. READ SENT. NO.nnnn VARIABLE (X. . . X) NOT ALPHA TYP-IN CORR	Variable X. . . X is not alphabetic.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the correct form of the variable.
23. READ SENT. NO.nnnn INCORRECT ITEM SIZE TYPE-IN CORR	The variables of the READ sentence make up an incorrect item size. The user may type in any larger acceptable item size. He must be sure, however, that his input data items are padded to this size. If the Data Conversion routine is used to prepare the data, the items will be padded to the next larger acceptable size.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the number of words including any padding of the prepared input item.
24. READ SENT. NO.nnnn FORMAT INCORRECT TYP-IN CORR	The sentence does not conform to the format prescribed in the manual.	Type in the entire sentence, omitting the name of the sentence, one pseudo-word per Univac word. X. . . XΔΔΔΔΔΔΔ When finished type in a word of spaces.

	PRINT-OUT	DESCRIPTION	PROCEDURE
25.	READ-ITEM SENT. NO.nnnn VARIABLE (X. . . X) NOT ALPHA TYP-IN CORR	The variable x. . . x is not alphabetic.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the correct form of the variable.
26.	READ-ITEM SENT. NO.nnnn PSEUDO WORD MISSING TYP-IN CORR	Sentence number nnnn does not conform to the READ-ITEM format in that the pseudo word IF is missing and there appears to be additional information in the sentence, signifying the sentence was intended to be a READ-ITEM with an IF option.	Type in: IFΔΔΔΔΔΔΔΔΔΔ or ΔΔΔΔΔΔΔΔΔΔΔΔ If the IF option is not desired.
27.	READ-ITEM SENT. NO. nnnn VARIABLE MISSING TYP-IN CORR.	The variable of the subscripted array is missing.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the intended variable.
28.	READ-ITEM SENT. NO.nnnn PAREN MISSING (X. . . X) TYP-IN CORR	The parens are missing from the subscripted variable.	Type in: (1) (Variable)ΔΔΔΔ (2) (first subscript) Δ (3) (second subscript)Δ (4) (third subscript) Δ If there are no second and/or third subscripts type in a word of spaces for the third and fourth words of the four word type in.
29.	READ-ITEM SENT. NO. nnnn ARRAY SIZE ALPHABETIC (X. . . X) TYP-IN CORR	The size of the array is X. . . X and it is alphabetic. In a READ-ITEM sentence the array size must be numeric.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the array size.

PRINT-OUT	DESCRIPTION	PROCEDURE
30. READ-ITEM SENT. NO. nnnn MORE THAN 3 SUBSCRIPTS REWRITE	The array of sentence nnnn contains more than three subscripts.	Hit start bar to rewind tape;
		or
	See NOTE of Print-Out 1 of Phase 2 for an explanation of the breakpoint option.	Set breakpoint 2; force transfer to bypass the error.
31. READ-ITEM SENT. NO. nnnn ARRAY SIZE TOO LARGE REWRITE	There are more than 250 elements in the array listed in sentence number nnnn.	Hit start bar to rewind tapes;
		or
	See NOTE of Print-out 1 of Phase 2 for an explanation of the breakpoint option.	Set breakpoint 2; force transfer to bypass the error.
32. READ-ITEM SENT. NO. nnnn SENT REF MISSING TYP-IN CORR	The sentence number to which control is to be transferred if missing.	Type in: X. . . X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ where X. . . X is the sentence number to which control is to be transferred.
33. READ-ITEM SENT. NO. nnnn INCORRECT ITEM SIZE TYP-IN CORR	The item size of the array of sentence number nnnn is not an acceptable size. See manual for acceptable item sizes. The user may type in any larger acceptable item size. He must be sure, however, that his input data items are padded to this size. If the data conversion routine is used to prepare the data, the items will be padded to the next larger acceptable size.	Type in: X. . . X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ where X. . . X is the number of words including any padding of the prepared input.
34. SUBSCRIPT SENT. NO. nnnn VARIABLE (X. . . X) NOT ALPHA TYP-IN CORR	Variable X. . . X is not an alphabetic.	Type in: X. . . X$\underline{\Delta\Delta\Delta\Delta\Delta\Delta\Delta}$ where X. . . X is the correct form of the variable.

PRINT-OUT	DESCRIPTION	PROCEDURE
35. SUBSCRIPT SENT. NO. nnnn MORE THAN 3 SUBSCRIPTS REWRITE	The array contains more than three subscripts. See NOTE of Print-Out 1 of Phase 2 for an explanation of the break-point option.	Hit start bar to re-wind tapes; or Set breakpoint 2; force transfer to by-pass the error.
36. WRITE SENT. NO.nnnn VARIABLE (X. . . X) NOT ALPHA TYP-IN CORR	The variable X. . . X is not alpha-betic.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the correct form of the variable.
37. WRITE SENT NO. nnnn WORD AFTER (X. . . X) MISSING TYP-IN CORR	X. . . X will be either AND or FOR, and the word following either of these is missing. The omitted word will be either EDIT, CON-VERT, or UNIPRINTER.	Type in: (1) X. . . XΔΔΔΔΔΔΔ where X. . . X is the missing word or (2) ΔΔΔΔΔΔΔΔΔΔΔΔ If none of the options is desired.
38. WRITE SENT. NO. nnnn TOO MANY VARIABLES REWRITE	More than 50 variables are listed in the WRITE statement of sen-tence number nnnn. See NOTE of Print-Out 1 of Phase 2 for an explanation of the break-point option.	Hit start bar to re-wind tapes; or Set breakpoint 2; force transfer to by-pass the error
39. WRITE-ARRAY or WRITE-ITEM SENT. NO. nnnn VARIABLE (X. . . X) NOT ALPHA TYP-IN CORR	The variable X. . . X is not alpha-betic.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the correct form of the variable.

PRINT-OUT	DESCRIPTION	PROCEDURE
40. WRITE-ARRAY or WRITE-ITEM SENT. NO. nnnn WORD AFTER (X. . . X) MISSING TYP-IN CORR	X. . . X will be either AND or FOR, and the word following either of these is missing. The omitted word should be either EDIT, CONVERT, or UNIPRINTER	Type in: (1) X. . . XΔΔΔΔΔΔΔ where X. . . X is the missing word or (2) ΔΔΔΔΔΔΔΔΔΔΔΔ If none of the options is desired.
41. WRITE-ARRAY SENT. NO. nnnn VARIABLE MISSING TYP-IN CORR	The variable of the subscripted array is missing.	Type in: X. . . XΔΔΔΔΔΔ where X. . . X is the proper variable.
42. WRITE-ARRAY or WRITE-ITEM SENT. NO. nnnn PAREN MSNG TYP-IN CORR	A paren is missing from the subscripted array.	Type in: (1) (variable) ΔΔΔΔ (2) (first subscript) Δ (3) (second subscript) Δ (4) (third subscript) Δ If there are no secone and/or third subscripts type in a word of spaces for the third and fourth word of the four word type in.
43. WRITE-ARRAY or WRITE-ITEM SENT. NO. nnnn ARRAY SIZE (X. . . X) ALPHABETIC TYP-IN CORR	The size of the array is X. . . X and it is alphabetic. In a WRITE-ITEM sentence the array size must be numeric.	Type in: X. . . XΔΔΔΔΔΔΔΔ where X. . . X is the correct array size.

PRINT-OUT	DESCRIPTION	PROCEDURE
44. WRITE-ARRAY or WRITE-ITEM SENT. NO. nnnn MORE THAN 3 SUBSCRIPTS REWRITE	The array contains more than three subscripts. See NOTE of Print-Out 1 of Phase 2 for an explanation of the breakpoint option.	Hit start bar to rewind tapes; or Set breakpoint 2; force transfer to bypass the error.
45. WRITE-ARRAY or WRITE-ITEM SENT. NO. nnnn ARRAY SIZE TOO LARGE REWRITE	There are more than 250 elements of the array. See NOTE of Print-Out 1 of Phase 2 for an explanation of the breakpoint option.	Hit start bar to rewind tapes; or Set breakpoint 2; force transfer to bypass error.
46. VARY SENT. NO. nnnn NUMERIC VARIABLE (X. . . X) TYP-IN CORR	Sentence number nnnn is a VARY increment type sentence and X. . . X the variable is numeric instead of alphabetic.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the correct form of the variable.
47. VARY SENT. NO. nnnn SENT REF MISSING TYP-IN CORR	The range of sentences or sentence over which the VARY is to be executed is missing.	Type in: X. . . XΔΔΔΔΔΔΔ Y. . . YΔΔΔΔΔΔΔ where X. . . X is the first sentence number of the range and Y. . . Y the second. If the range consists of only one sentence type in ΔΔΔΔΔΔΔΔΔΔΔΔ for Y. . . Y the second sentence number.
48. VARY SENT. NO. nnnn	Sentence number nnnn is a VARY list type sentence and the var-	Hit start bar to rewind tapes;

PRINT-OUT	DESCRIPTION	PROCEDURE
ITEM SIZE INCORRECT REWRITE	iables have an incorrect number of values listed for them.	or Set breakpoint 2; force transfer to bypass the error.
49. VARY SENT. NO. nnnn VARBLE HAS ALPHA VALUE (X. . . X) TYP-IN CORR	Sentence number nnnn is a VARY list type sentence. The value X. . . X of one of the variables is alphabetic instead of numeric. A two word type in is provided for the correction.	Type in: (1) X. . . XΔΔΔΔΔΔΔ (2) (o̲)ooooooooeee where X. . . X is the base of the number and eee is its exponent. If the number is not in the power of ten notation type in a word of zeros for the second type in.
50. VARY SENT. NO. nnnn ABS VALUE NUMERIC (X. . . X) TYP-IN CORR	Absolute values may be taken of alphabetic variables when they appear in the pseudo code.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the correct form of the variable.
51. VARY SENT. NO. nnnn ABS SIGN MISSING (X. . . X)	At least two absolute signs are missing from sentence number nnnn. X. . . X should have been an absolute sign. However, if X. . . X is the pseudo word immediately following the missing absolute sign and if the user feels that other errors resulting from the missing sign will not develop, ... The quantity from which the absolute sign is missing will be considered as a variable for which the absolute value is to be taken.	Set breakpoint 2; force transfer to continue; otherwise; Hit start bar to rewind tapes and rewrite the sentence.

PRINT OUT	DESCRIPTION	PROCEDURE
1. SENT. REF (X. . . X) INCORRECT SENT. NO. nnnn TYP-IN CORR	Sentence number nnnn is a CLOSE-OUTPUT statement making reference to sentence number X. . . X, but X. . . X is not a WRITE statement.	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the sentence number of the intended WRITE statement.
2. VARIABLE (X. . . X) INCORRECT SENT. NO. nnnn TYP-IN CORR	Variable X. . . X has not been previously defined in any of the pseudo code sentences. Sentence number nnnn is a VARY statement X. . . X is either an alphabetic lower limit, increment, or upper limit. The reason for X. . . X's not being defined is due to an incorrectly spelled variable or unintentionally not defining the variable.	
	If X. . . X is misspelled and previously defined. . .	Type in: X. . . XΔΔΔΔΔΔΔ where X. . . X is the correct form of the variable.
	If X. . . X has not been previously defined, the error can only be corrected by rewriting the problem and defining the variable.	Type in: ΔΔΔΔΔΔΔΔΔΔΔΔ and the system will rewind all tapes.
3. SENT. NO. nnnn VARIABLE (X. . . X) NOT DEFINED TYP-IN CORR	Variable X. . . X is used as an argument either in sentence number nnnn or in the sentence following nnnn and has not been defined in any of the other pseudo code sentences. The variable can be defined by a three word type-in according to the following options.	

PRINT OUT	DESCRIPTION	PROCEDURE
3. cont.	(1) If the variable X. . . X is misspelled in sentence nnnn but defined elsewhere. . . .	Type in: X. . . XΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ where X. . . X is the correct spelling of the variable.
	(2) If X. . . X should be a numeric instead of a variable. . . .	Type in: Y. . . YΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ where Y. . . Y is the numeric
	(3) If X. . . X should be a numeric written as a power of ten.	Type in: POWΔOFΔTENΔΔ Y. . . YΔΔΔΔΔΔΔ (º) o o o o o e. . . e where e. . . e is the exponent.
	(4) If X. . . X is correctly spelled and not previously defined, but can be defined by the three word type-in. . . However, if it cannot be defined, type in three words of spaces and the system will rewind all tapes.	Type in: X. . . XΔΔΔΔΔΔΔ Y. . . YΔΔΔΔΔΔΔ ΔΔΔΔΔΔΔΔΔΔΔΔ or (º) oooooe. . . e if the numeric is a power of ten.
4. SENT. NO. nnnn TOO MANY INPUT OUTPUT SENTENCES REWRITE	There are more than 40 input, output and contain statements in the problem.	Hit start bar to rewind tapes.

PRINT OUT	DESCRIPTION	PROCEDURE
5. SENT. NO. nnnn ALL SERVOS PRE-ASSIGNED REWRITE	At least one servo must be left unassigned for the running program tape. Compilation can be continued by using the breakpoint option.	Hit start bar to rewind tapes; or Set breakpoint 3 ; force transfer to continue compilation.
6. SENT. NO.nnnn SENT. NO. mmmm 2 INPUTS (X. . . X)	Two input statements have been found for the same variable, X. . . X. mmmm and nnnn are the sentence numbers of the two input statements.	Hit start bar to rewind tapes; or Set breakpoint 3; force transfer to assign the same storage area.
7. SENT. NO. nnnn SENT. NO. mmmm 2 EDITED WRITE-ARRAYS (X. . . X)	The variable, X. . . X, has been mentioned in two write-array edit statements with contain statements. nnnn and mmmm are the sentence numbers of the respective statements.	Hit start bar to rewind tapes; or Set breakpoint 3; force transfer to assign the same storage and servos to both sentences.
8. SENT. NO. nnnn SENT. NO. mmmm WRNG PREREAD REWRITE	The same variable has been mentioned in two read-array statements, or in an input statement and a read-array. mmmm and nnnn are the numbers of the sentences involved in the error.	Hit start bar to rewind tapes; or Set breakpoint 3; force transfer to omit the read-array being processed.
9. SENT. NO. nnnn SENT. NO. mmmm OUTPUT ITEM TOO BIG	A write-array or write-item statement specifies an item size larger than that of the matching input or read-array statement listed as sentence number mmmm. This is not an error if there is another input area of sufficient size. If this is not the case, compilation must be terminated at the end of Phase 3.	Hit start bar to rewind tapes; or Set breakpoint 3; to continue processing the same statement.

	PRINT OUT	DESCRIPTION	PROCEDURE
10.	SENT. NO. nnnn SENT. NO. mmmm UNNECESSARY CONTAIN MAY BE OMITTED	A contain statement specifies the same variable listed in another statement, other than a write-array edit without a contain. nnnn and mmmm are the numbers of the sentences involved in the error.	Hit start bar to re-wind tapes, or Set breakpoing 3; force transfer to omit this statement.
11.	SENT. NO. nnnn TOO MANY SRVO TYP-IN CORR	All available servos have been assigned. If it is possible to double up on some servo a type in is provided for typing in the servo number. If, however, the remaining un-assigned servo which is being reserved for the running program tape is typed in the word TYP-IN CORR will be typed out again.	Type in: t∆∆∆∆∆∆∆∆∆∆∆ where t is the servo number or Type in: ∆∆∆∆∆∆∆∆∆∆∆∆ to rewind the tapes.
12.	SENT. NO.nnnn SENT. NO. mmmm 2 INPUT ADDRESSES FOR (X. . . X) REWRITE	There are two input addresses for the variable X. . . X. Where possible both sentence numbers nnnn and mmmm will be printed out. Compilation must be ter-minated at the end of Phase 3 if the breakpoint option is selected.	Hit start bar to rewind tapes, or Set breakpoint 3; for a transfer to omit the item being processed.

NOTE

Before selecting the breakpoint option provided with the error print outs of Phase 3, the user is urged to examine his error to make certain other errors will not result by continuing the compilation. It is very possible at this late stage of compilation to induce errors by continuing by means of the breakpoint option.

PHASE 4 ERROR PRINTOUTS

	PRINT OUT	DESCRIPTION	PROCEDURE
1.	STORG AREA (XΔΔΔΔΔΔΔΔΔΔΔ) NOT LISTED REWRITE	A computer or compiler section makes reference to storage area X, but area X is not mentioned anywhere in the pseudo coded sentences.	Hit start bar to rewind tapes;
2.	LABEL OF (COMPUTER-XXX) or (COMPILER-XXX) INCORRECT TYP-IN CORR	The label XXX of the computer or compiler sentence listed in the pseudo code does not match any of the labels of the COMPUTER or COMPILER SECTIONS	Type in: COMPUTER-XXX or COMPILER-XXX where XXX is the label of the intended COMPUTER or COMPILER section.
3.	SENT. mmmm IS REFERRED TO BUT IS NOT LISTED	A sentence of the type that makes reference to other sentences has made reference to sentence number mmmm, but mmmm was not used as a sentence number in the problem.	Hit start bar to rewind tapes.

SWEEP 1 ERROR PRINT OUTS

	PRINT OUT	DESCRIPTION	PROCEDURE
1.	IMPROPER CALL WORD (X. . . X) REWRITE	The call word, digits 1-3 of X. . . X appearing in the print out, cannot be found in the library catalog.	Hit start bar to rewind tapes.
2.	IMPROPER STG AREA IN (X. . . X) TYP-IN CORR	The storage area of X. . . X, a call word appearing in a COMPILER section, is not listed in the pseudo coded problem.	Type in: 000000000XXX where XXX is the relative address of the intended storage area.

3. IMPROPER PSEUDO WORD (X. . . X) REWRITE	The first digit of a word in a COMPILER section is numeric, but the word is not an RG or CN reference or a word of skips.	Hit start bar to rewind tapes.

ARITH-MATIC OR A-3 INPUT - OUTPUT ROUTINES

ERROR PRINT OUTS

PRINT-OUT	DESCRIPTION	PROCEDURE
1. WRNG SERVO (X. . . X) TYP-IN CORR	The servo digit, t, of the call word appearing in the print out is not 1 through 9 or (-) minus. X. . . X will be one of the following A-3 call words. GMSBBtxxxAAA, GMT. . . , or GMU. . . GASBBtxxxAAA, GAT . . . , or GAU . . . GMIBBxxxAAA GMNO($^{H}_{L}$)tsssAAA, GMO	Type in: The call word appearing in the print out but with the proper servo designation.
2. IMPROPER ADDRESS IN (X. . . X) TYP-IN CORR	The address part, AAA, of the call word appearing in the print out is an odd number, if one of the following A-3 call words are printed out as X. . . X: GMSBBtxxxAAA, GMT. . . , or GMU. . . GMNO($^{H}_{L}$)txxxAAA, GMO If GASBBtxxxAAA, GAT. . . , or GAU appear in the print-out, either digits BB are not divisible by 10 or AAA is odd. If GMIBBtxxxAAA appears in the print out, the absolute address for AAA is not divisible by 10.	Type in: The call word appearing in the print out but with the correct address.

3. 5TH DIGIT INCORRECT (X... X) TYP-IN CORR	The 5th digit, or density digit, is not an "H" indicating high density, or "L" indicating low density. X..X will be GMN0?tsssAAA or GMO...	Type in: The call word appearing in the print out but with the correct density digit.
4. IMPROPER ITEM SIZE (X... X) TYP-IN CORR	If the A-3 call word appearing on the print out is GMSBBtxxxAAA, GMT... or GMU... GASBBtxxxAAA, GAT... or GAU ... the item size, XXX, is an odd number. Item sizes must be even numbers. If the A-3 call word is a GMNO$\left(\begin{smallmatrix}H\\L\end{smallmatrix}\right)$tsssAAA or GMO..., the item size, sss, is in error either because xxx is odd, if sss is greater than or equal to 60; or because sss + AAA does not complete a block, if sss is greater than 60. If the A-3 call word is GMIBBtsssAAA, the item size sss is in error because it is not one of the allowable item sizes. See Chapter IV for allowable item sizes. If the A-3 call word is GMMA AAsssBB, item size sss is in error because it is either equal to zero, greater than 60, or an odd number.	Type in: The call word appearing in the print out but with the correct item size.
5. 8TH DIGIT INCORRECT (QZOAAAT? oooo) TYP-IN CORR	The 8th digit position of the QZO call word is neither W, R, or zero.	Type in: oooooooooo$\begin{matrix}(0)\\(R)\\(W)\end{matrix}$o

MATHEMATICAL ROUTINES ERROR PRINT OUTS

	PRINT OUTS	DESCRIPTION	PROCEDURE
1.	7 - 9th DIGIT INCORRECT (RNAAAAnnnBBB) or (GPNAAAnnnBBB) TYP-IN CORR	The exponent or root listed in the 7 to 9th digits of the call word appearing in the print out is in error because it is either less than 2 or is not numeric. The exponent or root nnn must be an integer greater than or equal to 2 or less than or equal to 999.	Type in: ooooooooonnn where nnn is the correct form of the exponent.

SYSTEM RESTRICTION PRINT OUTS

	PRINT-OUT	DESCRIPTION	PROCEDURE
1.	SYSTEM RESTRICTION 01 SENT. NO. nnnn REWRITE	Sentence number nnnn after being translated contains more than 220 words or entries for Sentence File 01. In most cases the sentence is an equation and can be rewritten as two smaller ones in order to correct this requirement.	Hit start bar to rewind tapes, or; Set breakpoint 1; force transfer to process the sentence following nnnn.
2.	SYSTEM RESTRICTION02 REWRITE	This printout will occur when the pseudo code sentences contain more than 100 control statements which alter the sequence of execution over a given sentence range. See NOTE of Print Out 1 of Phase 2 for an explanation of the breakpoint option.	Hit start bar to rewind tapes, or; Set breakpoint 2; force transfer to bypass the error.
3.	SYSTEM RESTRICTION 03 SENT. NO. nnnn REWRITE	Sentence number nnnn is an equation that contains more than 100 operations. Re-examine the equation to see if it can be written as two smaller ones. See NOTE of Print Out 1 of Phase 2 for an explanation of the breakpoint option.	Hit start bar to rewind tapes, or; Set breakpoint 2; force transfer to bypass the error.

4.	SYSTEM RESTRICTION 04 SENT. NO.nnnn REWRITE	Sentence number nnnn is an equation containing more than 30 redundant operations. The equation must be regrouped or written as two smaller equations.	Hit start bar to rewind tapes.
5.	SYSTEM RESTRICTION 05 REWRITE	The problem in the present form contains too many control sentences, that is, too many sentences of the VARY, IF, JUMP, etc. types.	Hit start bar to rewind tapes.
6.	SYSTEM RESTRICTION 06 REWRITE	This print out occurs when a problem asks for an exceptional amount of storage area to be set aside for the running program tape. To correct this error, reexamine the problem to see if it is possible to double up on storage areas or rewrite the problem as two smaller ones.	Hit start bar to rewind tapes.
7.	SYSTEM RESTRICTION 07 REWRITE	During compilation portions of the memory are set aside to house the various files and lists produced as output from the phases ands sweeps. In the majority of cases these areas are sufficient in size to compile long complicated pseudo code problems. This print-out will result only for the exceptional problem. To correct this situation reexamine the problem to see if it is possible to rewrite the problem as two smaller ones and then begin a new compilation,	Hit start bar to rewind tapes.
8.	SYSTEM RESTRICTION 08 REWRITE	The problem contains more than 6 READ-ARRAY sentences.	Hit start bar to rewind tapes.

GENERAL DESCRIPTION OF ERROR PRINT-OUTS DURING THE PROBLEM RUN

Any error print out during the problem run is caused by one of the following two types of errors:

(1) The program is handling data which is outside the acceptable numerical range. This usually means alphabetic material is being incorrectly used as input data.

(2) The program is attempting to compute a function which is undefined or which is infinite, such as $\sqrt{-3}$ or log (-6).

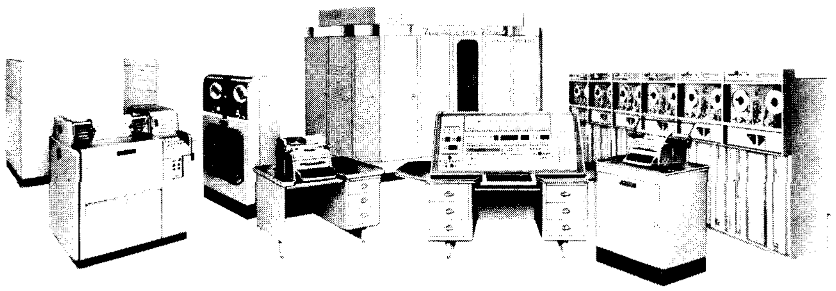
The first type of error will cause RUN ERR 01 to print out on the Supervisory Control Printer or will cause the computer to stop on an "adder alph." The second type of error will cause one of the print-outs from RUN ERR 02 through RUN ERR 07.

By carefully examining the RECORD EDIT, the programmer can identify the variables involved in the calculation causing the error, and by following the procedure listed with the print-out, the programmer can by-pass the error. This action is recommended only when the programmer fully understands the error and is certain that bypassing the error will not harm the results of the run. Ordinarily, the programmer should take a memory dump to detect the error and correct it by rewriting the necessary portion of his pseudo coded problem. For details on using the RECORD EDIT in by passing running tape errors, see the "MATH-MATIC PROGRAMMER'S MANUAL."

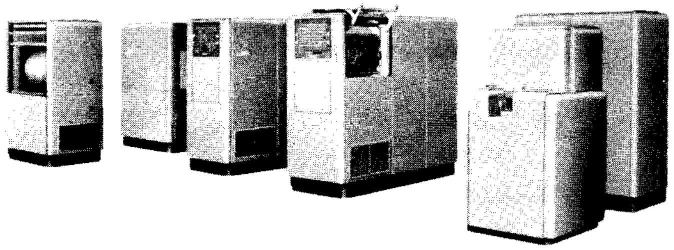
In RUN ERR 01, RUN ERR 02, and RUN ERR 03 the XXXX in the second word of the printout is the initial address of the next operation on the program tape. If the start bar is hit, 000000U0XXXX is performed and the error is by passed. In RUN ERR 04 through RUN ERR 07 the XXXX of the left hand instruction printed out in the second word of the print-out is the first address of the two word floating decimal input which caused the error. By typing in a proper input value to lines XXXX and XXXX + 1 and hitting the start bar, the operation in which the error occurred will be performed again.

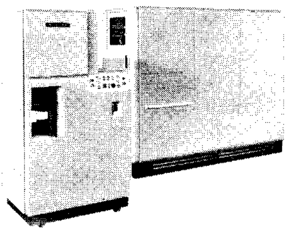
RUNNING PROGRAM TAPE ERROR PRINT-OUTS

	PRINT-OUT	DESCRIPTION	PROCEDURE
1.	RUN ERR 01 (ooooooUoxxxx)	A quantity produced by the floating decimal Arithmetic has an exponent with 12 significant digits.	Hit start bar to by pass the operation.
2.	RUN ERR 02 (ooooooUoxxxx)	Division by zero is being attempted.	Hit start bar to by pass the operation.
3.	RUN ERR 03 (ooooooUoxxxx)	Calculation of tangent A is being attempted for $A = \pi/2$, $3\pi/2$, etc. The result is infinite for these values.	Hit start bar; the program will continue with tangent $A = 10^{10}$.
4.	RUN ERR 04 (BoxxxxKooooo)	Calculation of arcsine A is being attempted for $\|A\| > 1$. The function is underfined in this range. A is in lines XXXX and XXXX + 1.	Type in a new value of A into XXXX and XXXX + 1; then hit the start bar to continue.
5.	RUN ERR 05 (BoxxxxKooooo)	Calculation of log A or $\log_{10} A$ for $A \leq 0$ is being attempted. The function is underfined in this range. A is in lines XXXX and XXXX + 1.	Type in a new value of A into lines XXXX and XXXX + 1; then hit the start bar to continue.
6.	RUN ERR 06 (BoxxxxKooooo)	Calculation of $\sqrt[n]{A}$ is being attempted for n even and $A < 0$. The function is imaginary in this range. A is in line XXXX and XXXX + 1.	Type in a new value of A into lines XXXX and XXXX + 1; then hit the start bar to continue.
7.	RUN ERR 07 (BoxxxxKooooo)	Calculation of $\sqrt{A}$ is being attempted for $A < 0$. The function is imaginary in this range. A is in XXXX and XXXX + 1.	Type in a new value of A into lines XXXX and XXXX + 1; then hit the start bar to continue.

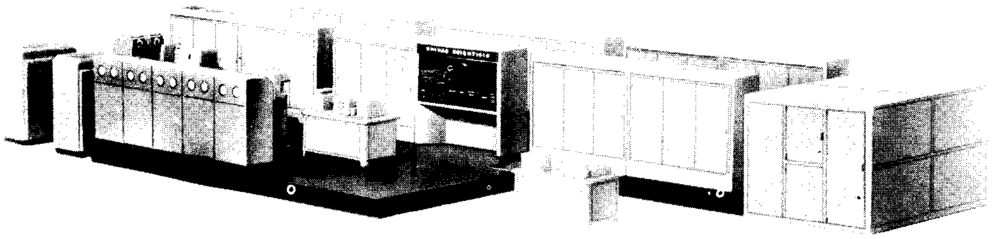

Univac II Systems • For data-automation which involves large volumes of input and output.

THE UNIVAC® FAMILY


Univac File-Computer • For instantaneous random access to large-scale internal storage—plus computation.


Univac 60 & 120 Computers • For speeding and simplifying the procedures of punched-card systems.

OF ELECTRONIC COMPUTERS


Univac Scientific Systems • For complex and intricate computations of engineering and research.

UNIVAC®

The FIRST Name in Electronic Computing Systems

1568 PRINTED IN U.S.A.