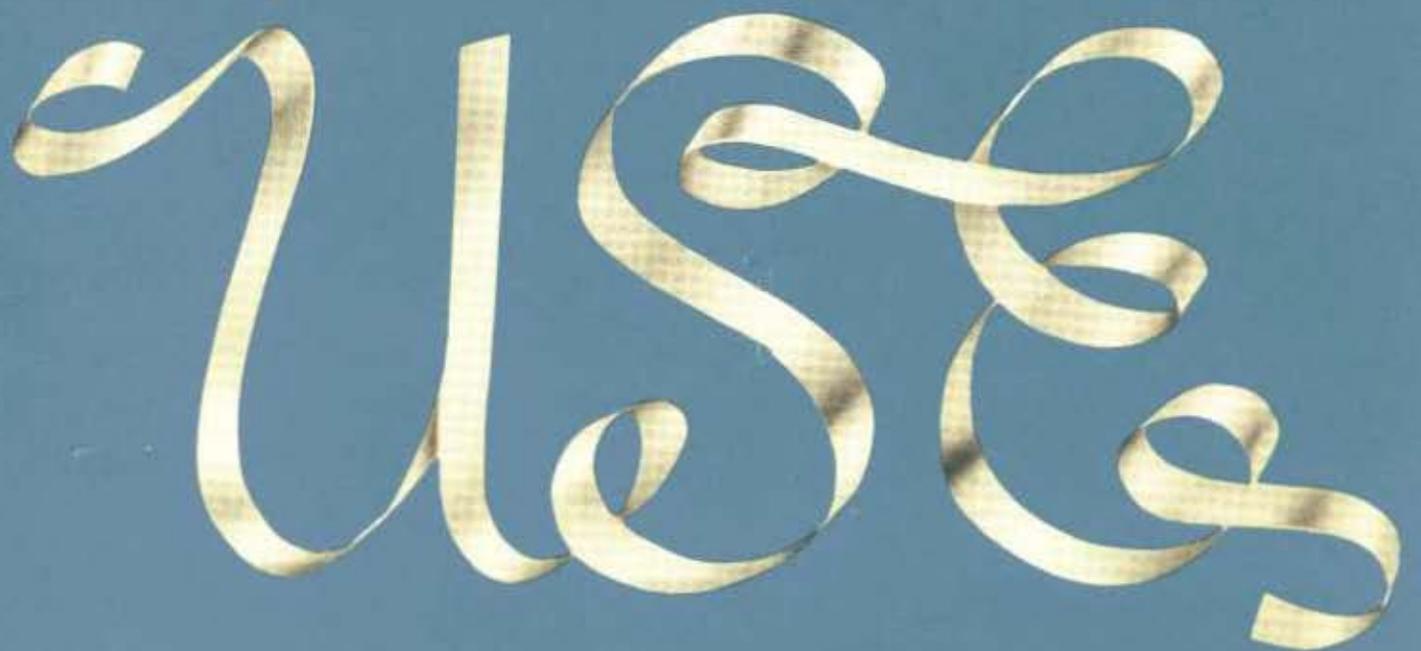


THE USE COMPILER PROGRAMMING MANUAL



THE USE COMPILER PROGRAMMING MANUAL

for the UNIVAC[®] SCIENTIFIC 1103A and 1105 Computers

univac scientific exchange

**THE USE COMPILER
PROGRAMMING MANUAL**

*for the UNIVAC[®] SCIENTIFIC
1103A and 1105 Computers*

Preface

The material contained in this manual is reproduced from that written at intervals by the members of the technical staff of the Ramo-Wooldridge Corporation under contract to Remington Rand. Some editing and additions have been made.

Contents

1. INTRODUCTION	i
I. THE USE COMPILER PROGRAMMING MANUAL FOR THE UNIVAC SCIENTIFIC 1103A	1
2. INPUT LANGUAGE	1
3. ITEM NUMBER FIELD	1
4. TAG FIELD	2
5. OPERATION CODE FIELD	3
5.1 1103A Instructions	3
5.2 Numbers	4
5.3 Subroutine Calling Sequences	4
5.4 Control Lines	4
6. ADDRESS FIELDS	5
6.1 1103A Instructions	5
6.2 Numbers	6
6.3 Subroutine Calling Sequences	6
6.4 Control Lines	6
7. COMMENTS FIELD	6
8. ASSIGNMENT OF MACHINE LOCATIONS	7
9. COMPILED REGION	7
9.1 Subroutine Temporary Pool	7
9.2 Constant Pool	7
9.3 Working Storage	7
9.4 Subroutines	7
10. EQUATING TAGS AND ITEM NUMBERS TO MACHINE ADDRESSES	8
11. PROGRAM STORAGE ON MAGNETIC TAPE	8
12. USE OF SUBROUTINES	9
13. FORM OF SUBROUTINES	11
14. CHANGES AND CORRECTIONS - RECOMPILING	12
15. USE OF CONTROL LINES	13
16. WARNINGS ISSUES BY THE COMPILER	17
TABLE 1	19
TABLE 2	20
TABLE 3	20
TABLE 4	21
TABLE 5	23
TABLE 6	24
II. TECHNICAL NOTES ON OPERATING THE USE COMPILER WITH THE UNIVAC SCIENTIFIC 1103A	26

Contents

(Continued)

III. THE USE COMPILER PROGRAMMING MANUAL FOR THE UNIVAC	
1105	40
1. <i>INTRODUCTION</i>	40
2. <i>INPUT LANGUAGE</i>	40
3. <i>ITEM NUMBER FIELD</i>	40
4. <i>TAG FIELD</i>	42
5. <i>OPERATION CODE FIELD</i>	42
5.1 <i>1105 Instructions</i>	42
5.2 <i>Numbers</i>	43
5.3 <i>Subroutine Calling Sequences</i>	43
5.4 <i>Control Lines</i>	44
6. <i>ADDRESS FIELDS</i>	44
6.1 <i>1105 Instructions</i>	44
6.2 <i>Numbers</i>	45
6.3 <i>Subroutine Calling Sequences</i>	45
6.4 <i>Control Lines</i>	46
7. <i>COMMENTS FIELD</i>	46
8. <i>ASSIGNMENT OF MACHINE LOCATIONS</i>	46
9. <i>COMPILED REGION</i>	46
9.1 <i>Subroutine Temporary Pool</i>	47
9.2 <i>Constant Pool</i>	47
9.3 <i>Working Storage</i>	47
9.4 <i>Subroutines</i>	47
10. <i>EQUATING TAGS TO MACHINE ADDRESSES</i>	47
11. <i>USE OF SUBROUTINES</i>	48
12. <i>FORM OF SUBROUTINES</i>	49
13. <i>CHANGES AND CORRECTIONS - RECOMPILING</i>	51
14. <i>USE OF CONTROL LINES</i>	52
15. <i>WARNINGS ISSUED BY THE COMPILER</i>	56
TABLE 1	57
TABLE 2	58
TABLE 3	58
TABLE 4	59
TABLE 5	61
IV. TECHNICAL NOTES ON OPERATING THE USE COMPILER WITH THE	
UNIVAC 1105	62
V. EXAMPLES DEMONSTRATING SUNDRY PROPERTIES OF THE	
COMPILER	73

1. INTRODUCTION

The purpose of the USE Compiler is to reduce the amount of effort required to prepare a correct code for the UNIVAC Scientific Computer, 1103A. The use of the compiler requires an understanding of the use of the 1103A and such an understanding is assumed in this manual.

Normally, the input to the compiler consists of a program written in the symbolic language described here and untyped on magnetic tape. In addition, a list of changes to be made to the main program may be supplied either on magnetic tape or cards. The output from the compiler is a tape which when listed on the off-line High-Speed Printer produces a side-by-side listing of the symbolic program as written by the programmer along with the translated machine language program in octal. This tape may be used to load the program into the computer. In addition a loadable binary tape may be produced. — There are three existing versions of the USE Compiler:

USE-1 The original version of the USE Compiler. It produced the side by side format with the symbolic on the left and the octal equivalents on the right. The u, v, and comments fields were run together; that is, not in fixed columns in the output positions. Also compilation stopped each time an illegal item number was encountered.

USE-2 The output listing is revised from that of USE-1 in that the octal equivalents are on the left with the symbolic on the right. The u and v fields are in fixed columns with possible variation in case all or part is unusually long. The illegal item number stop no longer occurs. The letter O and the number 0 are treated as the same character (they were treated as different characters in the USE-1 version and this caused considerable difficulty). USE-2 has provision for a mid-point recovery stop. The standard USE interpretive operation codes are recognized and correctly translated. A special calling sequence is generated for subroutines with two inputs both of which are stored in the accumulator at the time of entry to the subroutine. The binary deck of this version was made available at the July, 1957 USE meeting.

USE-3 Includes the XS3-n feature which allows direct incorporation into a program of excess 3 coded characters suitable for direct printing. This version also handles relocatable binary subroutines — previous versions required subroutines to be in symbolic form. The binary deck of the USE-3 compiler was made available at the March, 1958 USE meeting.

In this section USE-3 is described.

I. The USE Compiler-Programming Manual for the UNIVAC Scientific 1103A

2. INPUT LANGUAGE:

The language accepted by the USE Compiler consists of a series of lines of coding. Most lines of coding produce either an instruction or a number in the final machine-language code. Some lines, however, generate more than one word in the final code, while others are directions to the compiler and do not appear at all in the machine-language code.

Each line of coding is terminated by a special "end of line" symbol "\$" or "-" which must appear nowhere except at the end of each line. A line contains up to six fields separated by commas. Characters to the left of the first comma are in the item number field. The tag field follows the first comma, the operation code field follows the second comma, the u address field follows the third comma, the v address field follows the fourth comma, and the comments field follows the fifth comma.

The number of characters in any field is not fixed. Spaces are *not* equivalent to zeros; in fact, spaces are ignored except in the comments field. In certain cases it is permissible to leave a field blank. However, the correct number of commas as stated above must precede each field which is present. Commas beyond the fifth in any line are simply characters within the comments field and have no special significance. The code may be written either on blank paper or on forms with the end of line symbols and certain of the commas pre-printed.

3. ITEM NUMBER FIELD:

An item number is a number greater than or equal to one, having at most six digits to the left of the decimal point and at most four digits to the right of the point. The compiler will associate an item number with each line of coding. Normally the item numbers will be computed in a sequential fashion by the compiler, but a programmer may override this automatic computation of item numbers and explicitly assign any item number he wishes to any line of coding. The only restriction is that the item number on any line (whether computed by the compiler or supplied by the programmer) must be greater than the item number on the preceding line. That is, the sequence of item numbers must increase monotonically throughout the entire program.

The compiler will obey the following rules in assigning item numbers:

1. If more than one digit appears in the item number field the number as it appears is taken as the item number of that line. If no decimal point appears the number is considered an integer. The position of the right-most digit becomes the "index position". Thus, the index position may be the one's position or any one of the four fractional digit positions permitted in an item number.

2. If exactly one digit appears in the item number field that digit is extracted into the index position of the last item number which was written out in full. If exactly one digit appears in the item number field of the first line of coding it is taken as the complete item number, and rule 1, above, applies.
3. If the item number field is blank a one is added into the index position of the item number associated with the immediately preceding line of coding. If the item number field of the first line of coding is blank item number 1 is assigned to that line and the ones position becomes the index position.

Note that the index position is changed only when more than one digit appears in the item number field.

The programmer may adopt any one of several schemes for the assignment of item numbers. For example:

Option 1. Write a complete item number explicitly on each line of coding. Any monotonically increasing sequence of item numbers may be used.

Option 2. Leave the item number field blank throughout the code. The compiler will follow rule 3 and simply assign the integral item numbers 1, 2, 3, . . . , to the successive lines of coding. If it is desired to have the compiler count up in steps of one-tenth or one-hundredth, simply write 1.0 or 1.00 in the item number field of the first line of coding. If it is desired to interrupt the sequence and begin again at a different number simply write the new starting number in full in the item number field of the appropriate line, (taking care, of course, that the item number written is larger than that which will be computed by the compiler for the immediately preceding line).

Option 3. Write out in full every tenth item number (namely, those whose final digit is 0), and on the nine intervening lines write the single digits 1, 2, . . . 9. A pre-printed form may be used on which the single digits are already printed in the item number fields and it is therefore necessary to make an entry in this field only on every tenth line. Note that it is not necessary to use every line of the pre-printed form since the typist may be instructed to ignore a line which contains only an item number.

The use of item numbers for making changes to a code, and as symbolic addresses, will be described later. The important points covered so far are these:

1. Each line of coding is assigned a unique item number.
2. It does not matter whether the line produces none, one, or more than one word in the final machine-language code.
3. The item numbers will form a monotonically increasing sequence throughout the program.

If the above rules produce an item number which is not greater than all preceding item numbers an error stop will occur and the program will not be compiled.

4. TAG FIELD:

A tag is a symbol whose principal purpose is to serve as a symbolic address for a cell in the computer. A tag may consist of from one to six alphanumeric characters (chosen from the letters A thru Z and the digits 0 thru 9) at least one of which is a letter.

Four particular tags are automatically equated to fixed machine addresses as follows:

TAG		Machine Address
FILL	Illegal machine address	30000)B
Q	The quotient register	31000)B
A	The accumulator	32000)B
D	The first drum address	40000)B

If a tag appears in the tag field of a line of coding which produces one or more words in the final machine language code, that tag will be equated to the machine address of the cell occupied by the word, or first word, so produced. Tags may also be equated to machine addresses in other ways which will be described later. (Sec. 10, 15.2, 15.3, 15.4, 15.5). It is neither necessary nor desirable to have an entry in the tag field of every line. On the other hand, each line which produces a word referred to by another word probably should be tagged.

5. OPERATION CODE FIELD:

The operation is a symbol appearing in the operation code field and containing one to six alphanumeric characters. There are four main classes of symbols which may be used in this field: Those which produce instructions in the final machine-language code, those which produce numbers, those which generate subroutine calling sequences, and those which permit the programmer to modify and control some of the functions of the compiler.

5.1 OPERATION SYMBOLS WHICH PRODUCE 1103A INSTRUCTIONS.

There are four types of symbols in this class.

a. Octal operation codes.

Any two octal digits will become the operation part (bits 30 thru 35) of an 1103A word.

b. Standard letter pair operation codes.

Any of the letter pair symbolic operation codes recommended by Remington Rand will be translated to the corresponding pair of octal digits and become the operation part of an 1103A word. These codes are listed in Table 1.

c. Certain three-character codes for 1103A j-type operations.

In each case these are formed by affixing a third character to one of the two-letter codes listed in Table 1. The presence of the third character does not affect the translation of the first two letters which become the operation part of an 1103A word. In addition the third character is translated to an octal digit and becomes the j digit (bits 27 thru 29) of the 1103A word. The permissible three character codes of this type are listed in Table 2.

d. Standard interpretive operation codes.

Any of the standard USE interpretive operation codes are translated so that the operation code is 14 (octal) and bits 24 thru 29 contain the pair of octal digits corresponding to the operation which is to be interpreted. These are listed in Table 6.

5.2 OPERATION SYMBOLS WHICH PRODUCE NUMBERS

There are four operation symbols which produce numbers in the final machine language code; "B", "F", "X", and XS3-n where n represents one of the first nine integers.

- a. The operation symbol "B", (possibly followed by up to five octal digits within the operation code field) introduces an octal integer which will become an 1103A binary integer in the machine language code.
- b. The operation symbol "F" introduces a generalized decimal number which will be translated to the corresponding 1103A floating point number.
- c. The operation "X" introduces a generalized decimal number which will be translated to the corresponding 1103A binary integer.

The exact form of a generalized decimal number is described in section 6.2.

- d. The operation XS3 or XS3-n introduces a series of characters (alphabetic, numeric and others), which will be translated to n words of six excess-three characters. XS3 is equivalent to XS3-1. The use of this operation is described more fully in section 6.2.

5.3 OPERATION SYMBOLS WHICH PRODUCE SUBROUTINE CALLING SEQUENCES

Operation symbols in this class are the names of subroutines. Two cases arise: internal subroutines which are directly available to the compiler as part of the machine library, and external subroutines which must be supplied along with the program in which they are to be used. The name of an internal subroutine may be any combination of one to six alphanumeric characters provided it is not identical to any other acceptable operation symbol. The name of an external subroutine must consist of one of the organization letter pairs listed in Table 3 followed by one to four alphanumeric characters. The appearance of the name of a subroutine in the operation code field will produce a calling sequence and, at a remote location, the subroutine itself. The calling sequence will transfer the appropriate parameters and arguments to the subroutine and return jump to it.

Depending on the number of parameters and arguments required, the calling sequence may consist of one, two, or three words in the final machine code. If none are required the calling sequence is simply a return jump instruction. If one parameter or argument is needed the return jump is preceded by a transmit positive instruction. For more than one the transmit positive instruction is preceded by a repeat instruction.

5.4 OPERATION SYMBOLS WHICH CONTROL THE COMPILER

The following are special control symbols which may appear in the operation code field:

"SETLOC"	"USES"	"ALARM"
"RESERV"	"CARDS"	"TEMPS"
"EQUALS"	"TAPE"	"INOUT"
"LOCATE"	"DELETE"	"Pn" where n represents
"COMPAT"	"NOMORE"	0, 1, . . . 9
"DUPx" where x represents V, U, UV, O, OV, OU, or OUV	"SUB"	"ENDSUB"
		"END"

Each of these control symbols is described in detail in section 15. In general these symbols do not produce words in the final machine code. Rather they tell the compiler how it should operate on the other lines of coding.

6. ADDRESS FIELDS

The entries which may appear in the u and v address fields are determined, for each line, by the operation symbol on that line.

6.1 THE ADDRESS FIELDS OF A LINE WHICH PRODUCES AN 1103A INSTRUCTION

The u and/or v address fields of an 1103A instruction line may contain any of the following types of terms:

- a. Decimal address – a decimal integer translated as the corresponding binary integer.
- b. Octal address – an octal integer followed by “)B” translated as the corresponding binary integer.
- c. Tag – one to six alphanumeric characters at least one of which is a letter, translated to the machine execution address to which it has been equated, as explained elsewhere (Sections 4 and 10).
- d. Item number (complete) – a decimal number including a decimal point, translated to the machine execution address to which it has been equated, as explained elsewhere (Section 10).
- e. Item number (fractional part only) – A decimal point followed by one to four digits in the u or v address field is taken to be an item number whose integer part is the same as the integer part of the item number associated with the line of coding in which it appears.
- f. Storage Address – A tag or an item number as described in paragraphs c, d, or e above followed by “)s” translated to the machine storage address to which it has been equated, as explained elsewhere (Sections 8 and 10).
- g. A constant pool address – written “L(---)” translated as the machine address, within the constant pool (Sec. 9) of the number which is enclosed by the parentheses. The number is written just as it would appear in the operation code field and the u and v address fields of a separate line of coding, but without any commas. That is, a “B”, “X”, or “F” followed by the significant digits and exponents if needed. (Sec. 6.2). If no “B”, “X”, or “F” appears, an “X” is assumed.
- h. Compound addresses – any series of up to seven of the types of terms described in paragraphs a through g above, separated by “+” or “-” signs, translated as the algebraic sum of the translations of the individual terms.
- i. A completely blank field will be translated to binary zero.
- j. Where the operation produces 4 octal digits as in the case of the interpretive operation codes (described in Section 5.1d), the v address field is translated as in an ordinary instruction. The u address field is translated in the ordinary way and then its absolute value is compared to 2^9 . If the value is more than 2^9 a warning is given.

6.2 THE ADDRESS FIELDS OF A LINE WHICH PRODUCES A NUMBER

- a. Octal integers – The octal digits in the address fields of a line containing the operation symbol “B” are taken together with any octal digits which may have followed the “B” within the operation code field as an octal integer and translated to the corresponding binary integer.
- b. A generalized decimal number, which may appear in the address fields of a line containing the operation symbols “X” or “F”, consists of a significant digits part and, if desired, a binary and/or decimal exponent part. The significant digits part of a generalized decimal number is simply a decimal number, preceded by a sign and including a decimal point if desired. The binary and/or decimal exponent parts, if present, follow the significant digits part. They are introduced by the letters “B” and “D” respectively and consist of decimal integers of no more than three digits preceded by a “+” or “-” sign if desired. The “+” sign may be omitted in both the significant digits part and also the exponent parts of the number.
- c. XS3-n – Normally, the desired characters are simply written in the u and v address fields. However, a space is represented by an asterisk and certain other special characters are represented by a character pair. The first character of such a pair is always #. The complete list of characters is given in Table 5. If the number of characters produced by the contents of the u and v fields is less than 6n, the n words will be filled out with space characters (01 octal) and a warning will be given; if greater than 6n, only the first 6n characters will be used and a warning will be given. If the character # is followed by a character not listed in Table 5, the # is ignored and a warning is given.

The characters required in the address fields of a line which produces a number may be written right across both the u and v fields. That is, the fourth comma of the line (which separates the u field from the v field) may occur before, among, or following the other characters. It will have no effect on the translation of the number.

6.3 THE ADDRESS FIELDS OF A LINE WHICH PRODUCES A SUBROUTINE CALLING SEQUENCE

The u address field of a line which produces a subroutine calling sequence is associated with the location of the subroutine arguments and/or parameters and may contain any of the types of terms permitted in the address fields of a line which produces an 1103A instruction. The v address field of a line which produces a subroutine calling sequence is associated with the location of the subroutine itself and must be blank or contain exactly one tag. The use of subroutines will be fully described later (Section 12).

6.4 THE ADDRESS FIELDS OF A COMPILER CONTROL LINE

The entries permitted in the address fields of a compiler control line depend on the particular control symbol which is being used in the operation code field and is described in detail as each control symbol is explained (Section 15). In general, they are either certain of the types of terms permitted in the address fields of a line which produces an 1103A instruction (Section 6.1) or symbols having special meaning in connection with a particular control function.

7. COMMENTS FIELD

The comments field is essentially ignored by the compiler but is reproduced on the output listing. The programmer may enter any comments he wishes except that the end of line symbol “\$” or

“-|” must occur only at the end, and the total number of characters between the third comma of the line (which separates the operation code field from the u address field) and the end of line symbol is limited to fifty-nine.

8. ASSIGNMENT OF MACHINE LOCATIONS

Each word in the machine language program will be associated with two machine locations:

1. A storage location at which the word is to be stored at the start of the program, and
2. An execution location which the word is expected to occupy when it is used in the program.

These two machine locations may be the same (and usually will be unless the program is large enough to require segmentation). In large, segmented programs the execution location may be a magnetic core address, while the storage location is on the magnetic drum but this is not required.

In order to associate these two machine locations with each word of the machine language program the compiler will maintain two location counters; one for the execution location and one for the storage location. Ordinarily both counters are set to four at the start of a compilation. As each word of the machine language program is produced it is associated with the addresses found in the counters, and the location counters are then advanced by one.

The programmer, however, may at any point in the program modify this sequential assignment of machine locations. By use of the RESERV control line one or both of the counters may be advanced by an arbitrary amount. The SETLOC control line completely interrupts the sequence and permits specification of new starting values for one or both of the location counters. It is also possible for the programmer to specify that certain parts of the program are to be stored on magnetic tape. This feature and the use of these control lines is fully described later in Sections 11, 15.1 and 15.2.

9. THE COMPILED REGION

In addition to instructions, numbers, and subroutine calling sequences which are produced directly as a result of the occurrence of the corresponding types of lines of coding, there will be a region of the final machine language code which is produced automatically by the compiler. This compiled region will contain, in general, four parts.

- 9.1 **THE SUBROUTINE TEMPORARY POOL** is a group of cells reserved for the use of all subroutines in the program which require common temporary storage. Its length depends on the requirements of the particular subroutines used in the program.
- 9.2 **THE CONSTANT POOL** is a group of cells each of which contains the binary translation of a number whose constant pool address has been referred to by the “L(. .)” notation somewhere in the program.
- 9.3 **THE WORKING STORAGE** is a group of cells each of whose addresses is equated to a tag used in the program but not otherwise equated to a machine location.
- 9.4 **THE SUBROUTINES** which are used in the program and not specifically located elsewhere by the programmer will be placed in the compiled region.

Ordinarily the execution and storage locations of the compiled region will be the same, and such that the compiled region will be located at the high numbered end of magnetic core storage. The programmer may, however, by use of the COMPAT(compile at) control line, cause the compiled region to be assigned arbitrary storage and execution locations. The COMPAT control line also permits the equating of a tag to the machine address of the compiled region. Thus the subroutine temporary pool may be referred to in the main program if desired. The use of the COMPAT control line is described more fully in Section 15.5.

10. EQUATING TAGS AND ITEM NUMBERS TO MACHINE ADDRESSES

As has been mentioned (Section 4), each tag which appears in the tag field of a line of coding which produces one or more words in the machine-language code is equated to the execution and storage locations associated with the word, or the first word, so produced. Thus a tag may appear on any line which produces an 1103A instruction, a number, or a subroutine calling sequence.

In most cases, the tag field of a control line should be left blank since control lines do not produce words in the machine-language code. There are four exceptions, however. The EQUALS control line equates the tag to arbitrary storage and execution addresses. The COMPAT control line equates the tag to the beginning of the compiled region (i.e., the beginning of the subroutine temporary pool). The RESERV control line equates the tag to the address of the first word of an unfilled region. The LOCATE control line equates the tag to the beginning of a subroutine. The use of these control lines is explained in detail in Sections 15.2, 15.3, 15.4 and 15.5. Any given tag may appear in the tag field only once in a program.

Tags which are not equated to machine addresses as a result of their occurrence in the tag field as explained above may appear in the u or v fields of lines which produce 1103A instructions or subroutine calling sequences. The special tags "A", "Q", "D", and "FILL" are equated to machine addresses as explained above (Sec. 4). Other tags are equated by the compiler to machine addresses within the compiled region as follows: those appearing in the v field of lines which produce subroutine calling sequences are equated to the addresses associated with the first word of that subroutine; those which appear in the u field of lines which produce subroutine calling sequences or in the u or v field of lines which produce 1103A instructions are equated to the addresses of cells in the working storage part of the compiled region.

Item numbers of lines of coding which could have a tag in the tag field may appear in the u or v fields of lines which produce 1103A instructions or in the u field of lines which produce subroutine calling sequences. Such item numbers are equated to the same machine addresses as a tag would be if it appeared in the tag field of that line.

11. PROGRAM STORAGE ON MAGNETIC TAPE

A program is considered to be made up of one or more segments separated by the occurrence of each "SETLOC" and "RESERV" control line (except "RESERV" within a subroutine) (Sections 15.1 and 15.2). By means of a special tape mode symbol "T(n)", (Where n represents a digit from 1 through 9), in the v address of a "SETLOC" control line the programmer may set the tape mode and designate a particular tape unit, n. This means that succeeding segments will be compiled for initial storage on the designated tape unit until a tape mode symbol specifying a different tape unit appears, or until a "SETLOC" line with something other than a tape mode symbol clears the tape mode.

The compiler automatically numbers in sequence, starting with one, the segments compiled for each tape unit as they occur in the program. When in the tape mode the storage location counter contains a fifteen bit "Tape Address" of which the high order four bits represent the tape unit

number and the remaining eleven bits, the segment number on that tape. Furthermore, when in the tape mode, the storage location counter is not advanced by one for each machine language word produced, but rather one for each segment. (The advancing of execution location counter is not affected by the tape mode.) Thus, the "Storage Location" associated with every word in a given tape mode segment will be the same and will be a composite fifteen bit number. This fifteen bit number could be the same as an internal machine address and it is up to the programmer to avoid using it incorrectly.

Normally, however, the programmer need not be concerned with the tape segment numbers. The segments of the program stored on tape will be completely identified so that a tape reading subroutine when supplied with a "Tape Address" as a parameter could locate and read in the desired segment. The programmer would supply the "Tape Address" by writing the tag associated with any word in the segment, followed by "s".

If it is desired to store the compiled region on tape initially, the tape mode symbol, "T(n)", may be written in the v address field of the "COMPAT" control line (Section 15.5). The compiled region will then become segment zero on the designated tape unit.

12. USE OF SUBROUTINES

The internal form of standard USE subroutines acceptable to the USE Compiler is explained in detail later in Section 13. To use a subroutine the following conventions must be understood:

- a. The subroutine is to be entered by a standard return jump instruction; the entrance is always the first word of the subroutine and the normal exit is always the third word.
- b. At the time of exit from the subroutine the results are always stored within the subroutine in the fourth and following cells. (They may, in addition to this, be stored in the accumulator or elsewhere).
- c. Previous to the time of entrance to the subroutine the arguments and/or parameters must have been transferred to specific cells within the subroutine. (Namely, the cells immediately following the cells reserved for results).
- d. The subroutine is self-contained (that is it includes its own instructions and constants, and refers only to itself) with the following exceptions:
 - 1) It may refer to the accumulator and to the Q-register.
 - 2) It may refer to the subroutine temporary pool in the compiled region (Section 9).
 - 3) It may refer to other standard USE subroutines.

Information, including the length of the subroutine, the number and order of results, possible location of results in addition to the standard location, number, order, and location required within the subroutine of parameters and/or arguments, number of cells used in the temporary pool, and other subroutines referred to, will be found in the individual subroutine write-up.

The use of a line of coding which produces a subroutine calling sequence will, in many cases, make it unnecessary for the programmer to concern himself with many of these details. Such a

line of coding will cause the subroutine itself to be placed either in the compiled region or at a position specified by the programmer. It will insure that a tag is equated to the machine locations associated with the first word of the subroutine. It will generate the coding in the main program necessary to transmit the arguments and/or parameters into the subroutine, and to return jump to the subroutine, and finally, unless otherwise specified by the programmer, it will insure that all subsidiary subroutines are included in the compiled region.

As has already been explained (Section 5.3) the presence of the name of a subroutine in the operation code field of a line of coding indicates that a calling sequence is to be generated. The v address field of the line may either be blank, or contain exactly one tag. If it contains a tag which is equated to machine locations as a result of its appearance in the tag field of some other line of coding, the subroutine will be placed at the storage address and prepared for execution at the execution address equated to that tag. It is up to the programmer to insure that there is room for the subroutine at the location specified. He may do this for example by appropriate use of control lines such as "RESERV" or "LOCATE", which are explained in detail later. If the subroutine is to be in a segment on magnetic tape a "LOCATE" control line must be used (Sections 15.2 and 15.4).

If the v address field of a line which produces a calling sequence contains a tag which does not appear in the tag field on any line of coding, or if it is blank, the subroutine will be placed in the compiled region and the tag, if any, will be equated to the machine locations, within the compiled region, associated with the first word of the subroutine.

The u address field of a line of coding which produces a subroutine calling sequence may contain any of the types of terms permitted in the address fields of a line which produces an 1103A instruction (Section 6.1). It represents the location of the argument or parameter (or the first of these if more than one is required). If two or more parameters and/or arguments are required they must be available, in the correct order, in sequential cells so that they may be transferred to the correct cells within the subroutine by a repeated transmit positive instruction which will be produced by the compiler.

The compiler will insure that any subsidiary subroutines which are referred to by the primary subroutine are included in the compiled region unless the programmer, by means of the USES control line, which is explained in detail later (Section 15.7), indicates that a copy of the subsidiary subroutine at some specified place outside of the compiled region is to be employed.

As was explained (Sec. 5.3), the calling sequence produced by the compiler will consist of either a single return jump; a transmit positive, return jump; or a repeat, transmit positive, return jump; depending on the number of parameters and/or arguments required. In any case the tag, if any, in the tag field of the line which produces the calling sequence is equated to the machine locations associated with the first word of the calling sequence.

It is possible to include a subroutine in a program without producing a calling sequence. This is accomplished by means of the LOCATE control line which specifically locates, at the point in the program where the LOCATE line occurs, the subroutine whose name appears in the u address field. The use of LOCATE is explained in detail in Section 15.4.

It is, of course, the programmer's responsibility to insure that every subroutine to which he refers (and any subsidiary subroutines which they require) are available to the compiler at the time his program is compiled. Any subroutine, provided it is in the proper form (described fully in Section 13), which is not in the internally stored library may be included with the coding for the main program. The compiler will accept such external subroutines, and in effect (though not physically) add them to the library while that one program is being compiled.

13. FORM OF SUBROUTINES

The compiler will compile subroutines which are in a magnetic tape internal library or included externally as part of the coding for the program. Both internal and external subroutines are written in the same form as other coding for the compiler with certain exceptions which are described here.

The item number field should be left blank on all lines of coding of a subroutine.

A subroutine consists, in general, of nine parts:

- | | |
|--------------------|------------------------------------|
| 1) Leading line | 6) Result cells |
| 2) Parameter lines | 7) Argument and/or parameter cells |
| 3) Entrance line | 8) Body of the routine |
| 4) Alarm exit | 9) Ending line |
| 5) Normal exit | |

- 1) The *leading line* of a subroutine must have the special control symbol "SUB" in the operation code field to indicate that a subroutine is to be processed. The u address field of the leading line contains the name of the subroutine, and the v address field contains an integer which is equal to the number of consecutive cells occupied by routine. The SUB control line is described fully in Section 15.12.
- 2) There are three types of *parameter lines* introduced, respectively, by the special control symbols "TEMPS", "INOUT", and "Pn" (where n represents a digit 0 through 9) in the operation code field.

A "TEMPS" control line is used to indicate, by an integer in the u address field, the number of consecutive cells used by the subroutine in the subroutine temporary pool in the compiled region. An integer in the v address field indicates the position within the temporary pool at which the temporaries for this subroutine are to start. Normally this is zero except when it is necessary to avoid conflict with a subsidiary subroutine which itself uses the temporary pool.

An "INOUT" control line is used to specify by means of integers in the u and v address fields, respectively, the number of arguments and/or parameters and the number of results. If either the "TEMPS" line or the "INOUT" line does not appear the corresponding parameters are taken to be zero.

Control lines introduced by "P0", "P1", . . . "P9" in the operation code field have no effect on the compilation of the program but are permitted to make USE subroutines compatible with other systems.

- 3) The *entrance line* produces the first word of the machine language subroutine. It should be an unconditional jump to the body of the subroutine.
- 4) The *alarm exit* line should have the control symbol "ALARM" in the operation code field. This produces the second word of the machine language subroutine. The compiler will supply an appropriate instruction for the cell. This will depend on the computation system in use at a particular installation and might, for example, be a return jump to an alarm print out routine located at a fixed position on the drum.
- 5) The *normal exit* line produces the third word in the machine language subroutine. It should be an unconditional jump instruction which may be set up by a return jump instruction in the main program.

6) *Result cells* and

7) *Parameter and/or argument cells* must be reserved within the subroutine immediately following the normal exit either by use of the "RESERV" control line or by making entries in the tag field on the correct number of lines.

8) The *body of the subroutine* may contain the following types of lines:

Lines which produce 1103A instructions
Lines which produce numbers
Lines which produce subroutine calling sequences
"EQUALS" and "RESERV" control lines

These lines may be written just as in other (non-subroutine) coding for the compiler except that:

- a. No item numbers may be used.
- b. No constant pool addresses of the form "L(. . .)" may be used.
- c. Tags which appear in the v address field of lines which produce subroutine calling sequences should not appear in the tag field. They are equated by the compiler to the machine locations associated with the first word of the subsidiary subroutine thus called for.
- d. Other tags which appear in the u or v address fields but not in the tag field are equated, in the order of their occurrence, to the addresses of cells in the subroutine temporary pool in the compiled region.

9) The *ending line* of a subroutine must contain the special control symbol "ENDSUB" in the operation code field.

14. CHANGES AND CORRECTIONS; RECOMPILING

The procedure described in this section in no way precludes any other method of correcting a program compiled by the *USE* Compiler. Since both a loadable binary tape and a side-by-side listing tape are produced as outputs from the compiler, the installation or individual programmer has complete freedom to proceed in whatever way is most advisable in each case. The binary tape may be loaded into the computer and binary corrections inserted manually or via any of the input media, or a tape editing routine may be used to correct the binary tape.

However, when it is desired to obtain the advantages of a re-compilation (such as an up-to-date listing, and freedom from patches) the compiler itself should be used. Since an output listing tape appears to the compiler to be the same, in all essential respects, as an original untyped main program tape, the compiler does not differentiate between the first and subsequent compilations of a program. Changes may be made at the time of the first compilation if desired.

Each program to be compiled may be accompanied by a list of changes either all on magnetic tape or all on cards. The changes are written exactly like other lines of coding except that the item numbers on successive lines need not be in order. However, where a single digit or a blank appears in the item number field of a change line, the compiler will supply an item number just as it does for lines of coding in the main program. The changes are then sorted by the compiler according to item number and merged with the lines of coding from the main program as compilation proceeds.

Three types of changes may be made: one-for-one replacements, insertions, and deletions. It is important to note that these are changes to the compiler language program, not to the machine language program, i.e., lines of coding, not computer words, are being replaced, inserted, and deleted. It is therefore entirely feasible to change control lines as well as lines which produce 1103A instructions or numbers. This is one of the advantages of re-compiling.

Replacements: If the item number of a change line is identical to the item number on a line of the main program, the change line will replace the line in the main program.

Insertions: If the item number of a change line is not identical to the item number on any line of the main program, the change line will be inserted between two lines in the main program at the point such that the resulting sequence of item numbers is monotonically increasing. This point is, of course, unique.

Deletions: If the first of a group of successive lines is replaced (as explained above) by a change line having the control symbol "DELETE" in the operation code field, the entire group of lines will be deleted from the main program. The extent of the group of lines to be deleted is specified in the address fields of the "DELETE" control line as is explained in detail in Section 15.10.

15. THE USE OF CONTROL LINES

Many of the special control symbols which may appear in the operation code field have already been mentioned. In this section the uses of all such control lines are explained in detail.

15.1 "SETLOC". The "SETLOC" control line normally sets the execution and storage location counters to the translated values of the expressions found in the u and v address fields respectively. These fields may contain any of the types of terms permitted in the u and v address fields of lines which produce 1103A instructions (Section 6.1) with three exceptions:

- 1) Constant pool addresses of the form "L(. . .)" are not allowed.
- 2) Item numbers are not allowed.
- 3) Any tags which are used must have appeared in the tag field on a preceding line of coding.

Alternatively, the v address field of a "SETLOC" control line may contain the single tape mode symbol "T(n)" to set the tape mode and designate tape unit n as the storage location of the succeeding segments.

Ordinarily if either the u or the v address field is blank the corresponding location counter is not altered (thus a blank is not equivalent to a zero in this case). However, if a "SETLOC" control line with a blank v address field occurs while the compiler is in the tape mode (Section 11) the storage location counter is increased by one to indicate a new segment in the same tape unit.

"SETLOC" is not allowed in subroutines. It should not have a tag in the tag field.

15.2 "RESERV". The "RESERV" control line normally advances the execution and storage location counters by the translated values of the expressions found in the u and v address fields respectively. These fields may contain the same types of terms as those described in the first paragraph of "SETLOC" above (Section 15.1).

Ordinarily if the u or v address field is blank the corresponding location counter is not altered. If a "RESERV" control line occurs while the compiler is in the tape mode (Sec. 11), the storage location counter is increased by one to indicate a new segment on the same tape unit regardless of the v address field.

If a tag appears in the tag field of a "RESERV" line, it is equated to the values found in the location counters before they are altered. Thus the tag is associated with the first cell in the reserved region, and, if in the tape mode, with the preceding segment.

Within a subroutine "RESERV" does not start a new segment. Also within a subroutine but not otherwise, the translated values of the contents of the u and v address fields must be the same and the reserved region is loaded with binary zeros.

- 15.3 "EQUALS". The "EQUALS" control line, which may appear anywhere in a program, equates the tag appearing in the tag field to the translated values of the expressions found in the u and v address fields. These fields may contain the same types of terms as those described in the first paragraph under "SETLOC" above (Sec. 15.1).

If either the u or the v address field (but not both) is blank, both the execution location and the storage location equated to the tag will be the same, and equal to the translated value of the expression in the non-blank address field.

- 15.4 "LOCATE". The "LOCATE" control line causes the subroutine whose name appears in the u address field to be included in the machine language code at the point where the "LOCATE" occurs. The v address field is not used and no calling sequence is produced.

If a tag appears in the tag field it is equated to the cell associated with the first word of the subroutine, i.e., the entrance.

The subroutine must be available to the compiler at the time the "LOCATE" occurs, either in the internal library or by having been included in a preceding part of the coding.

"LOCATE" may not be used within a subroutine.

- 15.5 "COMPAT". The "COMPAT" (compile at) control line permits the programmer to override the automatic placing of the compiled region at the high numbered end of core storage, and/or to equate a tag to the machine locations of the first cell of the compiled region (i.e., the beginning of the subroutine temporary pool). At most, one "COMPAT" control line may appear in a program but it may appear anywhere except within a subroutine.

Normally the execution and storage locations of the compiled region are made equal to the translated values of the expressions found in the u and v address fields, respectively. These fields may contain any of the types of terms described in the first paragraph under "SETLOC" above (Sec. 15.1). If the u address field is blank the execution location of the compiled region will be at the high numbered end of magnetic core storage. If the v address field is blank the storage location will be the same as the execution location.

Alternatively, the v address field of the "COMPAT" control line may contain the single tape mode symbol "T(n)" where n represents a digit 1 through 9. In this case the storage location of the compiled region will be segment zero on tape unit n.

If a tag appears in the tag field of a "COMPAT" control line, it is equated to the machine locations of the first cell of the compiled region.

15.6 "DUPx". The "DUPx" (duplicate) control line reduces the amount of writing required when the entries in corresponding fields on each of a group of successive lines would be identical. If the group of lines is immediately preceded by the appropriate "DUPx" control line, then, in all the lines of the group except the first, those fields which would be duplicates of corresponding fields in the first line may be left blank.

In writing the "DUPx" operation code the letters O, U, and V refer to the operation code field, and the u and v address fields respectively, and are used to indicate which of these fields are to be duplicated. Thus, the "DUPx" operation code symbol may take any of the following forms:

"DUPO"	"DUPOU"
"DUPU"	"DUPOV"
"DUPV"	"DUPUV"

The special case "DUPOUV", where all three of the fields are to be duplicated, is discussed below.

The extent of the group of lines on which the duplication is to be effective may be indicated in either one of two ways.

- 1) The u address field of the "DUPx" control line contains a single integer equal to the total number of lines in the group (including the first), and the v address field contains the word "TIMES", or
- 2) The u address field contains the word "THRU", and the v address field contains a single item number which is the same as that in the item number field on the last line of the group.

The first line following the "DUPx" line must be written out in full in the normal fashion. The remaining lines of the group are written in the normal way except that the duplicated fields are left blank.

The compiler actually fills in the blank fields. Therefore, the "DUPx" line itself will not appear on the output listing but each line of the group will appear in its complete form.

The special case, "DUPOUV", where all three fields are to be duplicated is treated in a slightly different way. The first line of the "group of identical lines" is written out in full immediately following the "DUPOUV" line as usual. But, the remaining lines of the group are not written at all. Therefore, only method 1 above for indicating the extent of the group is applicable in this case. The "DUPOUV" line and the following line will appear on the output listing. "DUPOUV" should not immediately precede "LOCATE", "SUB", "CARDS", or "TAPE" control lines.

"DUPx" may not be used within a subroutine.

15.7 "USES". As has been explained previously (Sections 12 and 15.4), the occurrence of a "LOCATE" control line, or of a line which generates a subroutine calling sequence, indicates that a specified subroutine is to be included in the program. If this primary subroutine requires subsidiary subroutines, the compiler, normally, will insure that such subsidiary subroutines are included in the compiled region and the primary subroutine will refer to them there. The programmer may, however, override this automatic inclusion of subsidiary subroutines by means of "USES" control lines following the line which specifies the primary subroutine (i.e., either a "LOCATE" control line, or a line which generates the subroutine calling sequence).

The u address field of the "USES" control line contains the name of a subroutine which is subsidiary to the most recently referenced primary subroutine. The v address field of the "USES" control line must contain exactly one tag. The compiler will then assume that the subsidiary subroutine whose name is in the u address field of the "USES" control line is available at the location specified by the tag in the v address field, and the primary subroutine will refer to the subroutine at that location. It is the *programmer's responsibility* to insure that the subsidiary subroutine is actually at the location indicated. This may be done, for example, by means of a "LOCATE" control line (Sec. 15.4).

A "USES" control line must not appear within a subroutine.

- 15.8 "CARDS". The "CARDS" control line indicates that the succeeding lines of coding are to be found on cards in the card reader. Since the output listing of the program will be entirely on magnetic tape this line will be omitted from the output. The cards themselves need be available only the first time the program is compiled.

The contents of the address fields of a "CARDS" control line is insignificant. This line must not appear within a subroutine.

- 15.9 "TAPE". The "TAPE" control line indicates that succeeding lines of coding are to be found on magnetic tape. This line would be the last line of a group which is on cards. It will not appear on the output listing since the entire output listing is on magnetic tape.

The contents of the address fields is insignificant on a "TAPE" control line. This line must not appear within a subroutine. (Of course, it may immediately follow a subroutine on cards.)

- 15.10 "DELETE". The control line "DELETE", which should appear only among the changes, is used to delete a group of consecutive lines from the main program. The extent of the group of lines to be deleted may be indicated in either one of two ways.

- 1) The u address of the "DELETE" line contains a single integer equal to the total number of lines in the group to be deleted, and the v address field contains either the word "LINE" or "LINES", or
- 2) The u address field contains the word "THRU", and the v address field contains a single item number which is the same as that in the item number field of the last line to be deleted.

If both the u and v address fields of the "DELETE" line are blank, the effect is the same as though "1, LINE" had been written there.

The compiler actually carries out the deletion and neither the "DELETE" line itself nor the deleted lines will appear in the output listing.

- 15.11 "NOMORE". The "NOMORE" control line must appear only as the last line of coding in the list of changes and indicates to the compiler that there are no more changes. The contents of the address fields are insignificant.

- 15.12 "SUB". The "SUB" control line may appear only as the leading line of a subroutine. The u address field contains the name of the subroutine. If the subroutine is in the internal machine library its name may consist of any combination of from one to six alphanumeric characters which does not conflict with any acceptable operation code symbol. If the subroutine is an external one sup-

plied along with the main program, its name must consist of one of the organization letter pairs listed in Table 3 followed by from one to four alphanumeric characters.

The v address field of a "SUB" control line contains an integer which is equal to the number of consecutive cells required by the subroutine. This includes cells for the entrance, alarm exit, normal exit, results, arguments and instructions, constants, and subsidiary subroutine calling sequences in the body of the routine, but does not include cells used in the subroutine temporary pool.

15.13 "ALARM". The "ALARM" control line should appear only immediately following the entrance line within a subroutine. It will produce an appropriate alarm exit instruction in the machine-language code. The contents of the address fields are insignificant.

15.14 "TEMPS". The "TEMPS" control line may appear at most once within each subroutine. It indicates to the compiler what part of the subroutine temporary pool in the compiled region is to be used by the subroutine in which it occurs. The u address field contains an integer which is equal to the number of consecutive cells used by this subroutine in the temporary pool. The v address field contains an integer which is equal to the number of cells to be left at the beginning of the temporary pool ahead of those used by this subroutine. These may be temporaries used by subsidiary subroutines. If either address field is blank, or if the "TEMPS" line does not appear within the subroutine, the corresponding quantities are taken to be zero.

15.15 "INOUT". The "INOUT" control line may appear at most once within each subroutine. The u address field contains an integer which is equal to the number of parameters and/or arguments which must be placed within the subroutine before it is entered. The v address field contains an integer which is equal to the number of results which the subroutine computes and stores within itself. If either address field is blank, or if the "INOUT" line does not appear within a subroutine, the corresponding quantities are taken to be zero.

15.16 "Pn". Control lines having "PO", "P1", . . . "P9", in the operation code field are reproduced on the output listing but have no effect on the program. They are permitted in order to make it possible for USE subroutines to be compatible with other systems which might require more parameters. They may occur only within subroutines.

15.17 "ENDSUB". The "ENDSUB" control line may appear only as the final line of each subroutine. The contents of the address fields is insignificant.

15.18 "END". The control line "END" must appear as the last line of the program. The contents of the u address field are insignificant. The v address field may contain any of the types of terms permitted in the address fields of lines which produce 1103A instructions. The translated value of the entry in the v address field will be stored in a particular location in the final block on the binary output tape. It may be used by the binary tape loading routine as the address at which computation is to begin.

16. WARNINGS ISSUED BY THE COMPILER

During compilation of a program the compiler may detect various situations which make it impossible to continue in the normal way. These situations may represent typing errors, peripheral or input equipment malfunction, programming errors or intentional use of certain features of the compiler in an unanticipated manner. In practically all such cases the compiler will take a prescribed action

to overcome the difficulty and then proceed with the compilation. A special warning symbol, "W", will be placed on the output listing on the line of coding in which the unusual situation was detected, and a footnote (referencing the item number of the line containing the warning symbol) will explain the reason for the warning. Table 4 lists some of the unusual situations which are detected and explains the action taken. In a very few cases the nature of the situation makes it impossible to continue at all.

TABLE I
STANDARD OPERATION CODES

FP	Floating Polynomial Multiply	01	EJ	Equality Jump	43
FI	Floating Inner Product	02	QJ	Q-Jump	44
UP	Unpack	03	MJ	Manually Selective Jump	45
NP	Normalize Pack	04	SJ	Sign Jump	46
NE	Normalize Exit	05	ZJ	Zero Jump	47
TP	Transmit Positive	11	QT	Q-controlled Transmit	51
TM	Transmit Magnitude	12	QA	Q-controlled Add	52
TN	Transmit Negative	13	QS	Q-controlled Substitute	53
IP	Interpret	14	LA	Left Shift in A	54
TU	Transmit u-address	15	LQ	Left Shift in Q	55
TV	Transmit v-address	16	MS	Manually Selective Stop	56
EF	External Function	17	PS	Program Stop	57
RA	Replace Add	21	PR	Print	61
LT	Left Transmit	22	PU	Punch	63
RS	Replace Subtract	23	FA	Floating Add	64
CC	Controlled Complement	27	FS	Floating Subtract	65
SP	Split Positive Entry	31	FM	Floating Multiply	66
SA	Split Add	32	FD	Floating Divide	67
SN	Split Negative Entry	33	MP	Multiply	71
SS	Split Subtract	34	MA	Multiply Add	72
AT	Add and Transmit	35	DV	Divide	73
ST	Subtract and Transmit	36	SF	Scale Factor	74
RJ	Return Jump	37	RP	Repeat	75
IJ	Index Jump	41	ER	External Read	76
TJ	Threshold Jump	42	EW	External Write	77

TABLE II

j-TYPE OPERATION CODES

NEO } NEN }	Normalize	050	PU1 } PU7 }	Punch 7 Levels	631
NE1 } NEQ }	Quasinormalize	051	RPO } RPN }	Modify neither u nor v	750
LTO } LTL }	Transmit A Left	220	RP1 } RPV }	Modify v	751
LT1 } LTR }	Transmit A Right	221	RP2 } RPU }	Modify u	752
MJO		450	RPB } RP3 }	Modify Both u and v	753
MJ1		451			
MJ2		452	ERO } ERA }	Read from IOA	760
MJ3		453	ER1 } ERB }	Read from IOB	761
MSO		560			
MS1		561	EWO } EWA }	Write to IOA	770
MS2		562	EW1 } EWB }	Write to IOB	771
MS3		563			
PUO } PU6 }	Punch 6 Levels	630			

TABLE III

PERMISSIBLE LEADING CHARACTERS FOR EXTERNAL SUBROUTINE NAMES

AP	Applied Physics Lab., John Hopkins University
BA	Boeing Airplane Company
CE	U.S. Army, Corps of Engineers
HO	Holloman Air Force Base
ML	Missiles Systems Division, Lockheed Aircraft Corp.
RR	Remington Rand Division, Sperry Rand Corp.
RW	Ramo-Wooldridge Corp.
WF	Wright Air Development Center

TABLE IV

WARNINGS ISSUED BY THE COMPILER

<i>WARNING</i>	<i>ACTION TAKEN</i>
Item Number illegal, too large, or out of order.	Compilation stops.
More than 6 characters in tag or operation code field.	Rightmost six characters are used.
More than 59 characters in u and v address fields.	Leftmost 59 characters are used.
More than 7 terms in u or v address fields.	Leftmost 7 terms are used.
Illegal term in u or v address fields.	Illegal term translated as zero.
Superfluous sign in u or v address fields.	Rightmost sign applied (i.e. sign closest to term).
Tag in u or v address fields which appeared more than once in tag field.	Location first equated to tag is used.
Item number used in u or v address field not equated to a machine location.	Item number translated as zero.
Constant pool full.	Address of form "L(. . .)" translated as zero.
Duplicate tag in tag field.	Location first equated to tag is used.
Tag incorrectly in tag field of control line.	Tag is ignored.
No room in directory for tag or item number.	Tag or item number will not be equated to a machine location.
Tag incorrectly used in u or v address fields of control line without having previously appeared in tag field.	Tag translated as zero.
Constant pool address of the form "L(. . .)" in u or v address field of control line.	Term translated as zero.
Translated value of expression in u or v address field $\geq 2^{15}$ or < 0 .	Used MOD 2^{15}
Translated value of expression in u address field $\geq 2^{12}$ or < 0 in j-type operation.	Used MOD 2^{12}
Illegal operation.	Translated to binary zero word.

TABLE IV (Continued)

WARNINGS ISSUED BY THE COMPILER

<i>WARNING</i>	<i>ACTION TAKEN</i>
Number contains too many digits, or an illegal character, or is too large or too small.	Number translated as binary zero.
No room in subroutine table, illegal name or external subroutine on subroutine not available to compiler.	Subroutine not included in program.
More than one "INOUT" or "TEMPS" control line in a subroutine.	Last one to appear is used.
Too many temps, arguments, or results used by subroutine.	Subroutine will not operate correctly.
Storage Regions overlap.	Will be loaded in order written with compiled region and subroutines last.
Location counter ≥ 7777 or 77777 .	Recycled to 0 or 40000.
"DUPOUV" applied to certain lines illegally.	"DUPOUV" is ignored.
Improper entries in u and/or v address fields of "DELETE" control line.	Treated as though it read "1, LINE".
More than one change with the same item number.	Last one used.
Both u and v address fields blank on an "EQUALS" control line.	Tag is equated to zero.
"USES" control line out of place.	Line ignored.
"COMPAT" used more than once in program.	First "COMPAT" applies.
Too many warnings	Further warnings not footnoted.

TABLE V

XS3 CHARACTER DESIRED	WRITE
Digits 0 thru 9	Digits 0 thru 9
Letters A thru Z except 0	Letters A thru Z except 0
., -, +,), (, /	., -, +,), (, /
Space	*
Letter O	#0
Fast Feed 1	#1
Fast Feed 2	#2
Fast Feed 3	#3
Fast Feed 4	#4
Asterisk *	#A
Breakpoint β	#B
Comma ,	#C
Dollar sign \$	#D
Multiline	#M
Number sign #	#N
Stop code	#S

TABLE VI
BASIC OPERATIONS

<i>OCTAL</i>	<i>MNEMONICS</i>	<i>FUNCTION</i>
00	LDR	Load Result Storage
01	LFR	Load F and R
02	ADD	Add
03	SBT	Subtract
04	MPY	Multiply
05	MPA	Multiply Add
06	PMP	Polynomial Multiply
07	DIV	Divide
10	SQR	Square Root
11	SIN	Sine
12	COS	Cosine
13	EXP	Exponential
14	LOG	Logarithm
15	ATN	Arctangent
20	STU	Store Unrounded
21	BTU	Block Transfer Unrounded
22	FIX	Floating to Stated
23	FLT	Stated to Floating
24	STR	Store Rounded
25	BTR	Block Transfer Rounded
77	NOP	No Operation
76	LBB	Locate B-Boxes
16	CIP	Conjugate Inner Product
17	NIP	Non-conjugate Inner Product
26	LDP	Load Positive
27	LDN	Load Negative
66	LDM	Load Magnitude
67	ADM	Add Magnitude

MATRIX OPERATIONS

<i>OCTAL</i>	<i>MNEMONICS</i>	<i>FUNCTION</i>
30	MADD	Add
31	MSBT	Subtract
32	MMPY	Multiply
33	MTRM	Transmit
34	MTRP	Transpose
35	MINV	Inverse
36	MSMP	Scalar Multiply
37	MDAD	Diagonal ADD
40	MSPR	Spur
41	MDET	Determinant
42	MROW	Row Select
43	MCOL	Column Select
44	MPAR	Partition
45	MRPL	Replace
47	MSET	Bank Set

TABLE VI (Continued)

DOUBLE PRECISION OPERATIONS

<i>OCTAL</i>	<i>MNEMONICS</i>	<i>FUNCTION</i>
50	DLDR	Load Result Storage
51	DLFR	Load F and R
52	DADD	Add
53	DSBT	Subtract
54	DMPY	Multiply
55	DMPA	Multiply Add
56	DPMP	Polynomial Multiply
57	DDIV	Divide
60	DSQR	Square Root
61	DSIN	Sine
62	DCOS	Cosine
63	DEXP	Exponential
64	DLOG	Logarithm
65	DATN	Arctangent
70	DSTU	Store Result Unrounded
71	DBTU	Block Transfer Unrounded
72	DFIX	Floating to Stated
73	DFLT	Stated to Floating
74	DSTR	Store Result Rounded
75	DBTR	Block Transfer Rounded

II. Technical Notes On Operating the USE Compiler with the UNIVAC Scientific 1103A

In order to make the USE Compiler compatible with the various operating systems used at the different installations, the beginning and end states have been defined well in toward the compiler. It is necessary for each installation to supply the coding needed to fit the compiler into its system.

The self-loading binary deck will load the compiler into the correct cells on the drum. Each installation may then dump it in any desired form on magnetic tape. The compiler does not read itself in from magnetic tape during compiling. It must be loaded onto the drum before control is transferred to it.

TO LOAD COMPILER FROM BINARY CARDS:

1. Place binary deck in reader. First three cards must be in correct order. Card with 12 punch in column 80 must be last.
2. Pick one read card.
3. Key in the following program, (octal).

00100	17	00000	00103
00101	75	30030	00000
00102	76	10000	00000
00103	40	00000	00005

4. Start at 00100 (octal). Computer reads in compiler and halts on an MSO.

At this point the basic compiler occupies cells 44200 through 63544 (octal). It may be dumped out on magnetic tape in any form desired, or used immediately to compile.

The loading process destroys the contents of cells 0 through 101 (octal).

TO COMPILE A PROGRAM:

1. The basic compiler must be stored in cells 44200 through 63544 (octal).
2. All magnetic tapes involved must be correctly positioned. (See Page 31)
3. Cards, if used, must be in hopper and the Bull cycled.

4. A correct set of input parameters must be in cells 44100 through 44137 (octal). (See Page 29)
5. Instructions to be executed following compilation should be in cells 44000 through 44077 (octal).
6. Start at 44200 (octal).

ON COMPLETION OF COMPILING:

1. Control is transferred to the instruction at 44000 (octal).
2. The output parameters have been set up in cells 44140 through 44177 (octal). (See Page 30)
3. As specified by the input parameters the magnetic tapes are either rewound or left in positions indicated in the output parameters.
4. The input parameters are unchanged.
5. The basic compiler is reset and may be used again without reloading.
6. The following cells have been used: 00000 through 07777 (octal), 63545 through 67777 (octal), and depending on the number of changes part of the range 70000 through 77777 (octal). (Precisely $17n$ (decimal) cells starting with 70000 (octal) will be used where n is the number of changes.)
7. The following cells are unchanged: 40000 through 43777 (octal) and the second and third banks of magnetic cores, 10000 through 27777 (octal).

SUMMARY OF COMPILER USE OF INTERNAL STORAGE

00000 – 07777	Entire first bank of cores will be destroyed
10000 – 17777	} Second and third bank of cores are not used
20000 – 27777	
40000 – 43777	Not used
44000 – 44077	Arbitrary program to follow compiling
44100 – 44137	Compiler input parameters
44140 – 44177	Compiler output parameters
44200 – 63544	The compiler
63545 – 67777	Used by compiler
70000 – 77777	17 n cells destroyed where n is the number of changes

COMPILER INPUT PARAMETERS

- 44100 Changes: 0 = None, 1 to 10 = Tape Unit, 12 = Cards
- 44101 Type of change tape: 0 = Unityper
- 44102 Main program: 1 to 10 = Tape Unit, 0 = Cards
- 44103 First line of main program: 0 = Tape, 12 = Cards
- 44104 Type of main program tape: 0 = Unityper, 1 = Output
- 44105 Tape unit for temporary: 1 to 10
- 44106 Tape unit for output: 1 to 10
- 44107 Tape unit for subroutine library: 1 to 10, 0 = None
- 44110 Number of blocks to beginning of library (in v)
- 44111 Tape unit for binary: 1 to 10, 0 = None
- 44112 Instruction for alarm line within subroutines
- 44113 Last address of core + 1 available to program being compiled (in u and v).
- 44114 Rewind main program tape: 0 = No, 1 = After first pass
- 44115 Rewind change tape: 0 = No, 1 = After first pass
- 44116 Rewind subr. library tape: 0 = No, 1 = After interlude
- 44117 Rewind binary tape: 0 = No, 1 = At end
- 44120 Rewind output tape: 0 = No, 1 = At end

- 44121 - 44137 Not yet assigned. Should be zero.

COMPILER OUTPUT PARAMETERS

44140 Reason for stopping: 0 = Normal completion
1 = Illegal input parameters
3 = Parity error
4 = TAPE command with no tape unit specified in input parameters.

44141 Number of blocks forward on tape unit 1
44142 " " " " 2
44143 " " " " 3
44144 " " " " 4
44145 " " " " 5
44146 " " " " 6
44147 " " " " 7
44150 " " " " 8
44151 " " " " 9
44152 " " " " 10

44153 Physical tape unit on which parity error occurred
44154 Logical tape unit on which parity error occurred:
0 = Change tape
1 = Main program
2 = Subroutine library tape
3 = Manuscript subroutine tape
4 = Temporary tape
5 = Binary tape
6 = Output tape

44155 Starting address following parity error
44156 - 44177: Net yet assigned

COMPILER USE OF MAGNETIC TAPES

The following logical tapes may be involved:

Symbolic output tape

Temporary (intermediate) tape

Binary output tape

Main program input tape

Change tape

Subroutine library tape

The symbolic output tape is also used by the compiler to hold external (manuscript) subroutines during the first pass and may be used as additional temporary working space if reprocessing the binary tape is necessary.

Each logical tape, if used at all, must be a unique physical tape with one exception. The changes and main program may be on the same tape if the main program immediately follows the last line of changes (i.e., the "no more" line). Each physical tape used must be on a unique unit with one exception. The binary tape may be on the same unit as the main program or the changes. In this event, the compiler at the appropriate time will rewind the main program or change tape and halt on an MS to 05050 (octal) so that a blank may be mounted. On restarting at 05050 (octal) compilation will continue. The binary tape will then be produced after the second pass.

The units for the symbolic output tape and for the temporary tape must always be specified in the input parameters.

The temporary tape must be a blank, (i.e., any information on it will be destroyed). All other tapes should be positioned at the point where writing or reading is to begin, with one exception. The beginning of the subroutine library may be any number of blocks (specified in the input parameters) from the initial position of the tape.

If any of the uniqueness rules for tape units are violated, or if the symbolic output and temporary tape units are not specified, compilation will not be attempted. Control will be transferred to cell 44000 (octal) immediately with the correct indication left in the output parameters.

If library tape is not specified, it is assumed that there is none. Any reference to subroutines not included as manuscript subroutines will cause warnings of "Illegal Operation" or "Subroutine not available."

If the main program starts on cards and includes a "Tape" line, and no main program tape has been specified, compilation will cease. Control will be transferred to cell 44000 with the correct indication left in the output parameters.

PREPARATION OF INPUT

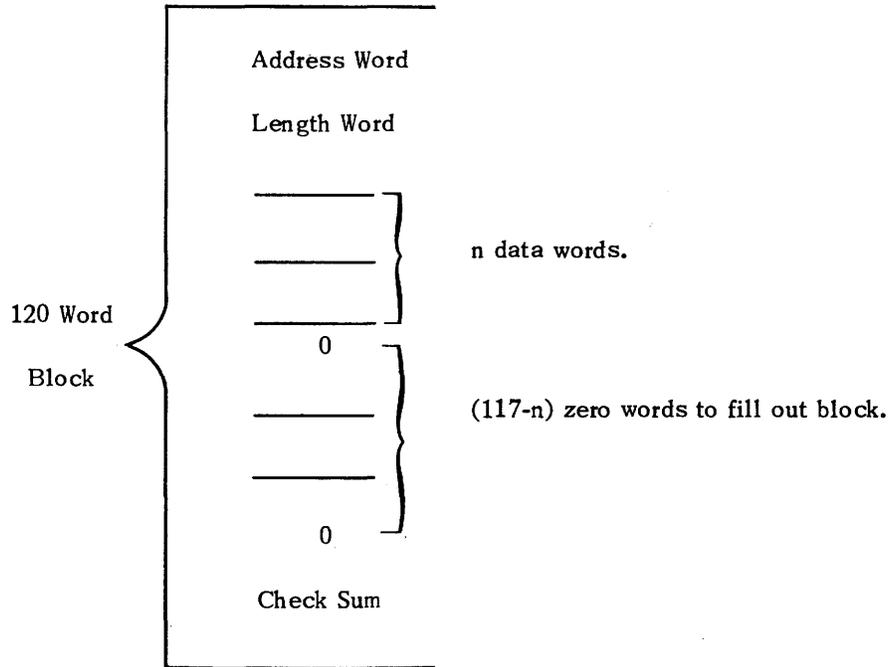
When reading changes and main program from either cards or untyped magnetic tape, the Compiler depends entirely on punctuation (commas and end-of-line symbols) to identify the various parts of the symbolic lines of coding. Card columns and character positions within the blockette have no significance. Spaces (except within comments) are completely ignored. Therefore, each installation may use any punching or untyping format it desires or simply place significant characters one after another with no spacing at all. It is not necessary to begin each line of coding on a new card or a new blockette.

USE COMPILER CHARACTER CODES

CHARACTER	OCTAL CODE	CARD ROWS	CHARACTER	OCTAL CODE	CARD ROWS
0	03	0	M	47	4, 11
1	04	1	N	50	5, 11
2	05	2	O	51	6, 11
3	06	3	P	52	7, 11
4	07	4	Q	53	8, 11
5	10	5	R	54	9, 11
6	11	6	S	65	2, 0
7	12	7	T	66	3, 0
8	13	8	U	67	4, 0
9	14	9	V	70	5, 0
A	24	1, 12	W	71	6, 0
B	25	2, 12	X	72	7, 0
C	26	3, 12	Y	73	8, 0
D	27	4, 12	Z	74	9, 0
E	30	5, 12	(17	4, 8, 0
F	31	6, 12)	43	4, 8, 12
G	32	7, 12	.	22	3, 8, 12
H	33	8, 12	+	63	3, 8
I	34	9, 12	-	02	11
J	44	1, 11	SPACE	01	None
K	45	2, 11		21	3, 8, 0
L	46	3, 11	\$ END-OF-LINE	55	3, 8, 11

FORM OF BINARY TAPE

Each block of the binary tape produced by the compiler has the following form:



The OC (Operation Code) part of the address word may be either 00, introducing a normal block, or 40, introducing a "tape mode" block. A normal address word has the form:

00	EEEEEE	SSSSS	
6	15	15	bits

where EEEEE is the execution address, and SSSSS is the storage address of the first data word. A tape mode address word has the form:

40	EEEEEE	T	S	
6	15	4	11	bits

where EEEEE is the execution address, T specifies a magnetic tape unit ($1 \leq T \leq 10$), and S is a sequence number. The sequence number S associated with a particular tape number T in the address word of a tape mode block is always larger than any other sequence number associated with the same tape number in the address word of any preceding block, i.e., the sequence numbers associated with a given tape number must increase throughout the entire binary tape.

Normally, the length word takes the following form:

00	NNNNN	00000
----	-------	-------

The u address part of the length word, NNNNN, contains n, the number of data words in the block. [n ≤ 117 (decimal); usually n = 117.]

The length word in the last block takes the form:

40 0000Q BBBBB

where the v address part, BBBBB, is the "begin computing address" (i.e., the translated value of the v field of the END line of coding). There are no data words in the last block.

The n data words immediately follow the length word. Following the data words are as many binary zero words as needed to fill out the block save one word.

The check sum, which is always the last word of the block, is the right 36 bits of the sum of the split extensions of all other words in the block. That is, the address word, the length word, and the n data words are included in the check sum.

TO LOAD A BINARY TAPE PRODUCED BY THE COMPILER

Although not a part of the basic compiler, a routine is provided which will load binary tapes produced by the compiler. If the binary cards for the binary tape loader are included with the compiler binary deck, the routine will be read into cells 70000 thru 70425 (octal) when the compiler is loaded. It may be dumped out in any convenient form. Note that the binary tape loader will be destroyed if compiling with changes is done; it should, therefore, be saved on magnetic tape.

The binary tape loader reads in the binary tape produced by the USE Compiler and stores the data words of each block from the tape into consecutive locations starting at the storage address specified by the address word in the block. The loader stores input data into cores; onto the Magnetic Drum, other than the core image (76000 - 77777 (octal)); or onto magnetic tapes other than the binary tape itself or those specially restricted.

The loader puts an MJ into the Operation Code portion of cell 00000 (octal), i.e., F₁, unless data being read in from the binary tape specifies otherwise.

To start, the Uniservo number on which the binary tape is located is inserted into Q, and then the machine is started at 70000 (octal). The contents of cells 00000 - 02000 (octal) are transferred to the core image, and the routine transfers itself into the first portion of core storage.

The routine, after getting into core storage, sums the constant portion of itself to ensure that it is intact. If the sum check fails, the Flexowriter prints an E and the machine comes to an MSO stop. If the machine is started again at this point, the routine will proceed exactly as if the sum had checked.

The loader checks the parity and sum of every block read in; reads forward and backward using the three different biases if necessary; and, if either check fails in each of all six cases, prints out on the Flexowriter MAGNETIC TAPE BLOCK READ IN CHECK FAILS followed by the storage address of the first word of the block to be stored. This address may consist of the tape unit number in decimal and the segment number in octal or simply a machine address in octal. The machine then comes to an MSO stop. If the machine is then started the block as last read in (reading backward at low bias) is transferred exactly as if both checks had worked after the last read in.

The loader checks any tape unit number to which a block is to be transferred against the tape unit number on which the binary tape is located and any other tape units which are not to be written on and whose numbers have been inserted into cells 70407 thru 70417 inclusive. If the tape unit number is illegal, the Flexowriter prints ILLEGAL TAPE UNIT NUMBER followed by the tape unit number in decimal. The binary tape unit number is put into register Q, the core image is restored, and the machine stops at MSO 70000) B.

If the non-tape storage address of any data word is illegal, i.e., between 10000 and 37777 octal or above 76000 octal, the Flexowriter prints ILLEGAL STORAGE ADDRESS followed by the first such address corresponding to a data word in the block.* All data words in the block which have legal addresses are properly stored. The binary tape unit number is then put into register Q, the core image is restored, and the machine stops at MSO 70000) B.

After the last block on the binary tape has been read in, the core image is restored and the machine stops at an MSO to the starting address of the program which has been read in.

**Note to permit loading 2 or 3 banks of cores cell 70370 (octal) should be changed to*

00 00000 20001 or 00 00000 30001

respectively.

USE COMPILER BINARY SUBROUTINE TAPE

This section applies also to the UNIVAC 1105 USE compiler.

The USE Compiler is independent of the subroutine library with which it is used. That is, changes may be made to the library without changing the compiler itself. This is achieved by maintaining an index on the library tape, which is read by the compiler at the appropriate time.

Subroutines on the library tape are in a relocatable binary form. In general, they are stored as though they were to be located at 4000)B and as though the temporary pool (compiled region) were to be located at 7000)B. Once the actual locations of the subroutine and the compiled region have been determined for a particular program, the addresses in the library form of the subroutine can be properly modified.

Those addresses less than 4000)B are constants, those between 4000)B and 7000)B are relative to the location of the subroutine, and those greater than 7000)B are in the temporary pool. Certain exceptions occur in this scheme, principally constants whose address parts are greater than 4000)B, and references to subsidiary subroutines. These exceptions are handled by introducing special key words into the subroutine to indicate how the exceptional cases are to be modified.

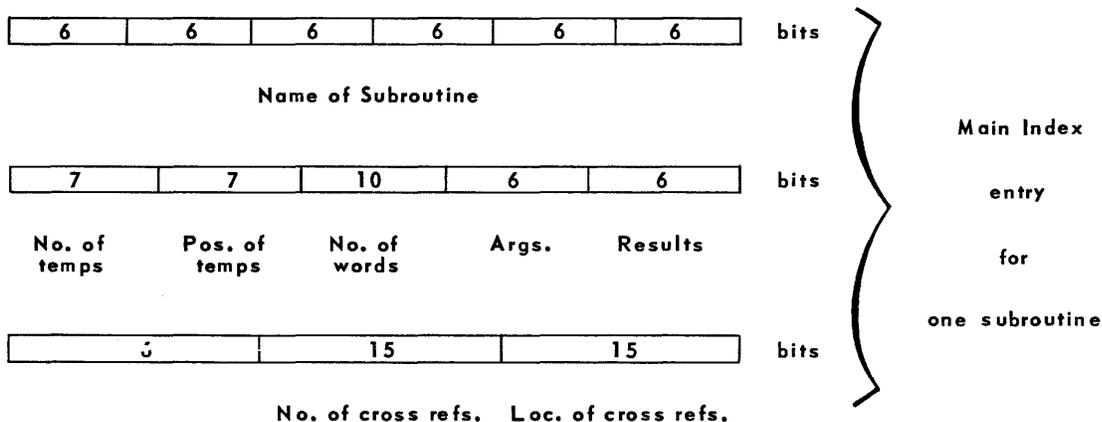
The following is a detailed description of the precise form of the library tape.

A USE Subroutine Library consists of a main index, a cross reference index and any number (≤ 120) of subroutines.

The library need not start at the beginning of the tape. The main index, the cross reference index, and the first subroutine must each start a new block.

The Main Index – The main index has an entry for each subroutine in the library. The entries in the index are in the same order as the subroutines are on the tape.

Each entry requires three computer words as follows:



The first word of the entry consists of the name of the subroutine in six six-bit excess-three characters.

The second word of the entry contains five numbers starting from the left: seven bits give the number of cells to be reserved for this routine in the subroutine temporary pool, seven bits give

the number of cells to be reserved at the beginning of the subroutine temporary pool ahead of those reserved for this routine, ten bits give the number of consecutive cells required for the subroutine, six bits give the number of arguments to be transferred into the subroutine by the calling sequence, and the rightmost six bits give the number of results produced by the subroutine. The information contained in the first two words of the entry is the same as that in the u and v addresses of the "SUB", "TEMPS", and "INOUT" lines of the subroutine.

The third word of the entry contains two numbers concerning subsidiary subroutines. The u address portion of the third word of the entry contains the number of different subroutines referred to (by means of calling sequence generator lines) in the primary subroutine. The v address portion of the third word of the entry gives the relative position (starting with zero) in the cross reference index of the first word associated with this primary subroutine.

The first word following the final entry in the main index is the flag word 77 00000 00000.

The Cross Reference Index: The cross reference index begins at the beginning of the block which immediately follows the block containing the flag word which ends the main index.

The cross reference index consists of a series of groups of subroutine names, each given as six six-bit excess-three characters. Each group contains the names of all subsidiary subroutines referred to by one of the primary subroutines. If a subroutine listed in the main index does not refer to any other subroutine there will be no group in the cross reference index corresponding to it. Within each group the subsidiary subroutine names must be in the same order as their first appearances in the operation code field of the primary subroutine.

Immediately following the final name in the final group in the cross reference index is the flag word 77 00000 00000. Immediately following this flag word is a single word check sum of all words in both the main index and the cross reference index with the exception of the two flag words.

The Library Subroutines: The first word of the first subroutine in library form is the first word in the block immediately following the block containing the check sum which ends the cross reference index. Each subroutine immediately follows its predecessor (i.e., only the first subroutine is required to start a new block) except that the last word in each block is a single word check sum of the other 119 words in that block.

In library form a subroutine consists of an identification word, base words which when properly modified become the words of the compiled subroutine, and usually some key words which indicate certain types of modification.

Identification Word: The first word in a subroutine in library form is always an identification word of the form:

57 55371 N

The 57 55371 pattern serves as a check indicating the beginning of a subroutine. The N is the total number of words in the subroutine on tape. N includes the identification word, all base words, and all key words, but does not include any block check sum words.

Base Words: Each base word is of the form OP, B, B where the values of its address parts, B, depend on the type of modification required in order to convert them into the address parts, A, of the word in the compiled subroutine. (The operation part of a base word is unmodified.)

Each address, A, (i.e., the u and the v part of each word) in a compiled subroutine must be expressible as the sum (modulo 2^{15}) of a constant, C, and at most one of the following variable locations:*

- E_s , the execution address of the beginning of the subroutine.
- S_s , the storage address of the beginning of the subroutine.
- E_T , the execution address of the beginning of the temporary pool.
- S_T , the storage address of the beginning of the temporary pool.
- X_i ($i = 0, 1, 2, \dots$), the execution address of the beginning of the i^{th} subsidiary subroutine.

An address is called a "normal address" if it is expressible in the form C , $C + E_s$, or $C + E_T$, and if the low order four octal digits of C (C modulo 2^{12}) are in the ranges shown in line 2 of Table 1. If B is then defined as shown in line 3 it is clear that the value of B determines the form of the address, A, which may be computed as shown in line 5.

If an address is not normal, B is defined as shown in line 2 of Table 2. In these cases, the value of B does not determine the form of the address so 10-bit codes shown in line 3 are assigned to distinguish the various cases. The address, A, in the compiled subroutine may then be computed as shown in line 4.

If both addresses in a base word are normal and if the operation part of the word is not 70, the base word is normal. If one or both of the addresses in a base word are not normal, or if the operation part of the word is 70, the base word is abnormal.

As has been pointed out above, normal base words can be properly modified without reference to key words. Groups of one or more consecutive abnormal base words may be interspersed among the normal words. Each group of abnormal words is preceded by a key word of the following form:

70	u code	v code	m
6 bits	10 bits	10 bits	10 bits

The 70 in the operation part of a key word serves to identify it as such and to distinguish it from normal words. The number of abnormal words in the group is given by m. The u code and v code parts of the key word specify the type of modification required for the u and v address parts of the abnormal words in the group. The exact form of these 10-bit codes is given in line 3, Table 2.

The first base word (the entrance) of a subroutine must be normal (e.g., MJO, O, BODY). The second base word is a dummy word which is arbitrarily replaced with an "Alam" line. A key word may not appear before these first two base words.

*Technically this restriction prohibits an address which is dependent on two or more of the above variable locations although such an address is legal in a symbolic subroutine. It is felt that this added restriction is of no practical consequence. Note, for example, that the difference of two addresses within the subroutine is expressible as a constant.

TABLE 1

1. Form of expression for normal address	$A = C$	$A = C + E_s$	$A = C + E_T$
2. Range of $C \bmod 2^{12}$	[0, 3777]	[0, 2777]	[0, 0777]
3. Definition of B	$B = C$	$B = C + 4000$	$B = C + 7000$
4. Range of $B \bmod 2^{12}$	[0, 3777]	[4000, 6777]	[7000, 7777]
5. To compute address	$A = B$	$A = B + E_s - 4000$	$A = B + E_T - 7000$

TABLE 2

1. Form of expression for abnormal address	C	$A = C + E_s$	$A = C + S_s$	$A = C + E_T$	$A = C + S_T$	$A = C + X_i$
2. Definition of B	$B = C$	$B = C + 4000$	$B = C + 4000$	$B = C + 7000$	$B = C + 7000$	$B = C + 2000$
3. 10-bit code	0001	0002	0003	0004	0005	1000+i
4. To compute address	$A = B$	$A = B + E_s - 4000$	$A = B + S_s - 4000$	$A = B + E_T - 7000$	$A = B + S_T - 7000$	$A = B + X_i - 2000$

III. The USE Compiler-Programming Manual for the UNIVAC 1105

1. INTRODUCTION

The purpose of the USE Compiler is to reduce the amount of effort required to prepare a correct code for the UNIVAC Scientific Computer, 1105. The use of the compiler requires an understanding of the use of 1105 computing system, and such an understanding is assumed in this manual.

Normally, the input to the compiler consists of a program written in the symbolic language described here and untyped on magnetic tape. In addition a list of changes to be made to the main program may be supplied on magnetic tape. The output from the compiler is a tape which when listed on the offline High-Speed Printer produces a side-by-side listing of the symbolic program as written by the programmer along with the translated machine language program in octal. In addition a loadable binary tape is produced.

2. INPUT LANGUAGE

The language accepted by the USE Compiler consists of a series of lines of coding. Most lines of coding produce either an instruction or a number in the final machine-language code. Some lines, however, generate more than one word in the final code, while others are directions to the compiler and do not appear at all in the machine-language code.

Each line of coding is terminated by a special "end of line" symbol "\$" or "-" which must appear nowhere except at the end of a line. A line contains up to six fields separated by commas ", ". Characters to the left of the first comma are in the item number field. The tag field follows the first comma, the operation code field follows the second comma, the u address field follows the third comma, the v address field follows the fourth comma, and the comments field follows the fifth comma.

The number of characters in any field is not fixed. Spaces are *not* equivalent to zeros; in fact, spaces are ignored except in the comments field. In certain cases it is permissible to leave a field blank; however, the correct number of commas as stated above must precede each field which is present. Commas beyond the fifth in any line are simply characters within the comments field and have no special significance. The code may be written either on blank paper or on forms with the end of line symbols and certain of the commas pre-printed.

3. ITEM NUMBER FIELD

An item number is a number greater than or equal to one, having at most six digits to the left of the decimal point and at most four digits to the right of the point. The compiler will associate

an item number with each line of coding. Normally the item numbers will be computed in a sequential fashion by the compiler, but a programmer may override this automatic computation of item numbers and explicitly assign any item number he wishes to any line of coding. The only restriction is that the item number on any line (whether computed by the compiler or supplied by the programmer) must be greater than the item number on the preceding line. That is, the sequence of item numbers must increase monotonically throughout the entire program.

The compiler will obey the following rules in assigning item numbers:

1. If more than one digit appears in the item number field the number as it appears is taken as the item number of that line. If no decimal point appears the number is considered an integer. The position of the right-most digit becomes the "index position". Thus, the index position may be the one's position or any one of the four fractional digit positions permitted in an item number.
2. If exactly one digit appears in the item number field that digit is extracted into the index position of the last item number which was written out in full. If exactly one digit appears in the item number field of the first line of coding it is taken as the complete item number, and rule 1, above, applies.
3. If the item number field is blank a one is added into the index position of the item number associated with the immediately preceding line of coding. If the item number field of the first line of coding is blank item number 1 is assigned to that line and the one's position becomes the index position.

Note that the index position is changed only when more than one digit appears in the item number field.

The programmer may adopt any one of several schemes for the assignment of item numbers. For example:

Option 1. Write a complete item number explicitly on each line of coding. Any monotonically increasing sequence of item numbers may be used.

Option 2. Leave the item number field blank throughout the code. The compiler will follow rule 3 and simply assign the integral item numbers 1, 2, 3, . . . , to the successive lines of coding. If it is desired to have the compiler count up in steps of one-tenth or one-hundredth, simply write 1.0 or 1.00 in the item number field of the first line of coding. If it is desired to interrupt the sequence and begin again at a different number simply write the new starting number in full in the item number field of the appropriate line, (taking care, of course, that the item number written is larger than that which will be computed by the compiler for the immediately preceding line).

Option 3. Write out in full every tenth item number (namely, those whose final digit is 0), and on the nine intervening lines write the single digits 1, 2, . . . 9. A pre-printed form may be used on which the single digits are already printed in the item number fields and it is therefore necessary to make an entry in this field only on every tenth line. Note that it is not necessary to use every line of the pre-printed form since the typist may be instructed to ignore a line which contains only an item number.

The use of item numbers for making changes to a code is described in Sec. 13. The important points covered so far are these:

1. Each line of coding is assigned a unique item number.
2. It does not matter whether the line produce none, one, or more than one word in the final machine-language code.
3. The item numbers will form a monotonically increasing sequence throughout the program.

If the above rules produce an item number which is not greater than all preceding item numbers an error will be indicated but compiling will continue.

4. TAG FIELD

A tag is a symbol whose principal purpose is to serve as a symbolic address for a cell in the computer. A tag may consist of from one to six alphanumeric characters (chosen from the letters A thru Z and the digits 0 thru 9) at least one of which is a letter. The letter "0" will be changed to a zero by the compiler.

Five particular tags are automatically equated to fixed machine addresses as follows:

<u>Tag</u>		<u>Machine Address</u>
FILL	Illegal machine address	30000)B
Q	The quotient register	31000)B
A	The accumulator	32000)B
D	The first drum address	40000)B
L	The present value of the location counters	

If a tag appears in the tag field of a line of coding which produces one or more words in the final machine-language code, that tag will be equated to the machine address of the cell occupied by the word, or first word, so produced. Tags may also be equated to machine addresses in other ways which are described in Sec. 10, 13.2, 14.3, 14.4, 14.5. It is neither necessary nor desirable to have an entry in the tag field of every line. On the other hand, each line which produces a word referred to by another word probably should be tagged.

5. OPERATION CODE FIELD

The operation is a symbol appearing in the operation code field and containing one to six alphanumeric characters. There are four main classes of symbols which may be used in this field: Those which produce instructions in the final machine language code, those which produce numbers, those which generate subroutine calling sequences, and those which permit the programmer to modify and control some of the functions of the compiler.

5.1 OPERATION SYMBOLS WHICH PRODUCE 1105 INSTRUCTIONS

There are three types of symbols in this class.

- a. Octal operation codes.

Any two octal digits will become the operation part (bits 30 thru 35) of an 1105 word.

- b. Standard letter pair operation codes.

Any of the letter pair symbolic operation codes recommended by Remington Rand will be translated to the corresponding pair of octal digits and become the operation part of an 1105 word. These codes are listed in Table 1.

- c. Certain three-character codes for the 1105 j-type operations.

In each case these are formed by affixing a third character to one of the two-letter codes listed in Table 1. The presence of the third character does not affect the translation of the first two letters which become the operation part of an 1105 word. In addition the third character is translated to an octal digit and becomes the j digit (bits 27 thru 29) of the 1105 word. The permissible three character codes of this type are listed in Table 2.

5.2 OPERATION SYMBOLS WHICH PRODUCE NUMBERS

There are four operation symbols which produce numbers in the final machine language code; "B", "F", "X", and XS3-n" where n represents one of the first nine integers.

- a. The operation symbol "B", possibly followed by up to five octal digits within the operation code field) introduces an octal integer which will become an 1105 binary integer in the machine language code.
- b. The operation symbol "F" introduces a generalized decimal number which will be translated to the corresponding 1105 floating point number.
- c. The operation "X" introduces a generalized decimal number which will be translated to the corresponding 1105 binary integer. The exact form of a generalized decimal number is described in section 6.2.
- d. The operation XS3 or XS3-n introduces a series of characters, (alphabetic, numeric and other) which will be translated to n words of six excess-three characters. XS3 is equivalent to XS3-1. The use of this operation is described fully in Section 6.2.

5.3 OPERATION SYMBOLS WHICH PRODUCE SUBROUTINE CALLING SEQUENCES

Operation symbols in this class are the names of subroutines. Two cases arise: internal subroutines which are directly available to the compiler as part of the machine library, and external subroutines which must be supplied along with the program in which they are to be used. The name of an internal subroutine may be any combination of one to six alphanumeric characters provided it is not identical to any other acceptable operation symbol. The name of an external subroutine must consist of an organization letter pair such as those listed in Table 3 followed by one to four alphanumeric characters. The appearance of the name of a subroutine in the operation code field will produce a calling sequence and, at a remote location, the subroutine itself. The calling sequence will transfer the appropriate parameters and arguments to the subroutine and return jump to it.

Depending on the number of parameters and arguments required, the calling sequence may consist of one, two, or three words in the final machine code. If none are required the calling sequence is simply a return jump instruction. If one parameter or argument is needed the return

jump is preceded by a transmit positive instruction. For more than one, the transmit positive instruction is preceded by a repeat instruction. If there are two arguments, both of which are in the accumulator, the return jump is preceded by a left transmit and a transmit positive instruction.

5.4 OPERATION SYMBOLS WHICH CONTROL THE COMPILER

The following are special control symbols which may appear in the operation code field:

"SETLOC"	"USES"	"ALARM"
"RESERV"	"CARDS"	"TEMPS"
"EQUALS"	"TAPE"	"INOUT"
"LOCATE"	"DELETE"	"Pn" where n represents
"COMPAT"	"NOMORE"	0, 1, . . . 9
"DUPx" where x represents	"SUB"	"ENDSUB"
V, U, UV, O, OV,		"END"
OU, or OUV		

Each of these control symbols is described in detail in Sec. 14. In general they do not produce words in the final machine code. Rather they tell the compiler how it should operate on the other lines of coding.

6. ADDRESS FIELDS

The entries which may appear in the u and v address fields are determined, for each line, by the operation symbol on that line.

6.1 THE ADDRESS FIELDS OF A LINE WHICH PRODUCES AN 1105 INSTRUCTION

The u and/or v address fields of an 1105 instruction line may contain any of the following types of terms.

- a. Decimal address – a decimal integer translated as the corresponding binary integer.
- b. Octal address – an octal integer followed by "(B)" translated as the corresponding binary integer.
- c. Tag-one to six alphanumeric characters at least one of which is a letter, translated to the machine execution address to which it has been equated as explained in Sections 4 and 10.

- d. Storage Address – A tag as described in paragraph c. above followed by “)S” translated to the machine storage address to which it has been equated as explained in Sections 8 and 10.
- e. A constant pool address – written “L(---)” translated as the machine address, within the constant pool (Sec. 9) of the number which is enclosed by the parentheses. The number is written just as it would appear in the operation code field and the u and v address fields of a separate line of coding, but without any commas. That is, a “B”, “X” or “F” followed by the significant digits and exponents if needed (Sec. 6.2). If no “B”, “X”, or “F” appears, an “X” is assumed.
- f. Compound addresses – any series of up to seven of the types of terms described in paragraphs a. through e. above, separated by “+” or “-” signs, translated as the algebraic sum of the translations of the individual terms.
- g. A completely blank field will be translated to binary zero.

6.2 THE ADDRESS FIELDS OF A LINE WHICH PRODUCES A NUMBER

- a. Octal integers – The octal digits in the address fields of a line containing the operation symbol “B” are taken together with any octal digits which may have followed the “B” within the operation code field as an octal integer and translated to the corresponding binary integer.
- b. A generalized decimal number which may appear in the address fields on a line containing the operation symbols “X” or “F”, consists of a significant digits part and, if desired, a binary and/or decimal exponent part. The significant digits part of a generalized decimal number is simply a decimal number, preceded by a sign and including a decimal point if desired. The binary and/or decimal exponent parts, if present, follow the significant digits part. They are introduced by the letters “B” and “D” respectively and consist of decimal integers of no more than three digits preceded by a “+” or “-” sign if desired. The “+” sign may be omitted in both the significant digits part and also the exponent parts of the number.
- c. XS3-n – Normally, the desired characters are simply written in the u and v address fields. However, a space is represented by an asterisk and certain other special characters are represented by a character pair. The first character of such a pair is always #. The complete list of characters is given in Table 5. If the number of characters produced by the contents of the u and v fields is less than 6n, the n words will be filled out with space characters (01 octal) and a warning will be given; if greater than 6n, only the first 6n characters will be used and a warning will be given. If the character # is followed by a character not listed in Table 5, the # is ignored and a warning is given.

The characters required in the address fields of a line which produces a number may be written right across both the u and v fields. That is, the fourth comma of the line (which separates the u field from the v field) may occur before, among, or following the other characters. It will have no effect on the translation of the number.

6.3 THE ADDRESS FIELDS OF A LINE WHICH PRODUCES A SUBROUTINE CALLING SEQUENCE

The u address field of a line which produces a subroutine calling sequence is associated with

the location of the subroutine arguments and/or parameters and may contain any of the types of terms permitted in the address fields of a line which produces an 1105 instruction. The v address field of a line which produces a subroutine calling sequence is associated with the location of the subroutine itself and must be blank or contain exactly one tag. The use of subroutines is described fully in Section 11.

6.4 THE ADDRESS FIELDS OF A COMPILER CONTROL LINE

The entries permitted in the address fields of a compiler control line depend on the particular control symbol which is being used in the operation code field, and will be fully described later as each control symbol is explained in Section 14. In general, they are either certain of the types of terms permitted in the address fields of a line which produces an 1105 instruction (Sec. 6.1) or symbols having special meaning in connection with a particular control function.

7. COMMENTS FIELD

The comments field is essentially ignored by the compiler but is reproduced on the output listing. The programmer may enter any comments he wishes except that the end of line symbol "\$" or "-|" must occur only at the end, and the total number of characters between the third comma of the line (which separates the operation code field from the u address field) and the end of line symbol is limited to fifty-nine.

8. ASSIGNMENT OF MACHINE LOCATIONS

Each word in the machine language program will be associated with two machine locations:

1. A storage location at which the word is to be stored at the start of the program, and
2. An execution location which the word is expected to occupy when it is used in the program.

These two machine locations may be the same (and usually will be unless the program is large enough to require segmentation). In large segmented programs, the execution location may be a magnetic core address, while the storage location is on the magnetic drum but this is not required.

In order to associate these two machine locations with each word of the machine language program the compiler will maintain two location counters; one for the execution and one for the storage location. Ordinarily both counters are set to four at the start of a compilation. As each word of the machine language program is produced it is associated with the addresses found in the counters, and the location counters are then advanced by one.

The programmer, however, may at any point in the program modify this sequential assignment of machine locations. By use of the RESERV control line one or both of the counters may be advanced by an arbitrary amount. The SETLOC control line completely interrupts the sequence and permits specification of new starting values for one or both of the location counters. The use of these control lines is described fully in Sections 14.1 and 14.2.

9. THE COMPILED REGION

In addition to instructions, numbers, and subroutine calling sequences which are produced

directly as a result of the occurrence of the corresponding types of lines of coding, there will be a region of the final machine language code which is produced automatically by the compiler. This compiled region will contain, in general, four parts.

- 9.1 **THE SUBROUTINE TEMPORARY POOL** is a group of cells reserved for the use of all subroutines in the program which require common temporary storage. Its length depends on the requirements of the particular subroutines used in the program.
- 9.2 **THE CONSTANT POOL** is a group of cells each of which contains the binary translation of a number whose constant pool address has been referred to by the "L(. .)" notation somewhere in the program.
- 9.3 **THE WORKING STORAGE** is a group of cells each of whose addresses is equated to a tag used in the program but not otherwise equated to a machine location.
- 9.4 **THE SUBROUTINES** which are used in the program and not specifically located elsewhere by the programmer will be placed in the compiled region.

Ordinarily the execution and storage locations of the compiled region will be the same, and such that the compiled region will be located at the high numbered end of magnetic core storage. The programmer may, however, by use of the COMPAT (compile at) control line, cause the compiled region to be assigned arbitrary storage and execution locations. The COMPAT control line also permits the equating of a tag to the machine address of the compiled region. Thus the subroutine temporary pool may be referred to in the main program if desired. The use of the COMPAT control line is described fully in Section 14.5.

10. EQUATING TAGS TO MACHINE ADDRESSES

As has been mentioned in Sec. 4 each tag which appears in the tag field of a line of coding which produces one or more words in the machine-language code is equated to the execution and storage locations associated with the word, or the first word, so produced. Thus a tag may appear on any line which produces an 1105 instruction, a number, or a subroutine calling sequence.

In most cases, the tag field of a control line should be left blank since control lines do not produce words in the machine-language code. There are four exceptions, however. The EQUALS control line equates the tag to arbitrary storage and execution addresses. The COMPAT control line equates the tag to the beginning of the compiled region (i.e. the beginning of the subroutine temporary pool). The RESERV control line equates the tag to the address of the first word of an unfilled region. The LOCATE control line equates the tag to the beginning of a subroutine. The use of these control lines is explained in detail in Sections 14.2, 14.3, 14.4 and 14.5. Any given tag may appear in the tag field only once in the main program and only once in any given subroutine.

Tags which are not equated to machine addresses as a result of their occurrence in the tag field as explained above may appear in the u or v fields of lines which produce 1105 instructions or subroutine calling sequences. The special tags "A", "Q", "D", "L", and "FILL" are equated to machine addresses. Other tags are equated by the compiler to machine addresses within the compiled region as follows: Those appearing in the v field of lines which produce subroutine

calling sequences are equated to the addresses associated with the first word of that subroutine. Those which appear in the u field of lines which produce subroutine calling sequences or in the u or v field of lines which produce 1105 instructions are equated to the addresses or cells in the working storage part of the compiled region.

11. USE OF SUBROUTINES

The internal form of standard USE subroutines acceptable to the USE compiler is explained in detail in Section 12. To use a subroutine the following conventions must be understood:

- a. The subroutine is to be entered by a standard return jump instruction; the entrance is always the first word of the subroutine and the normal exit is always the third word.
- b. At the time of exit from the subroutine the results are always stored within the subroutine in the fourth and following cells. (They may, in addition to this, be stored in the accumulator or elsewhere).
- c. Previous to the time of entrance to the subroutine the arguments and/or parameters must have been transferred to specific cells within the subroutine. (Namely, the cells immediately following the cells reserved for results.)
- d. The subroutine is self-contained (that is it includes its own instructions and constants, and refers only to itself) with the following exceptions:
 - 1) It may refer to the accumulator and to the Q-register.
 - 2) It may refer to the subroutine temporary pool in the compiled region (Section 9).
 - 3) It may refer to other standard USE subroutines.

Information, including the length of the subroutine, the number and order of results, possible location of results in addition to the standard location, number, order, and location required within the subroutine of parameters and/or arguments, number of cells used in the temporary pool, and other subroutines referred to, will be found in the individual subroutine write-up.

The use of a line of coding which produces a subroutine calling sequence will, in many cases, make it unnecessary for the programmer to concern himself with many of these details. Such a line of coding will cause the subroutine itself to be placed either in the compiled region (Sec. 9) or at a position specified by the programmer. It will insure that a tag is equated to the machine locations associated with the first word of the subroutine. It will generate the coding in the main program necessary to transmit the arguments and/or parameters into the subroutine, and finally, unless otherwise specified by the programmer, it will insure that all subsidiary subroutines are included in the compiled region.

As has already been explained (Sec. 5.3), the presence of the name of a subroutine in the operation code field of a line of coding indicates that a calling sequence is to be generated. The v address field of the line may either be blank, or contain exactly one tag. If it contains a tag which is equated to machine locations as a result of its appearance in the tag field of some other line of coding, the subroutine will be placed at the storage address and prepared for execution at

the execution address equated to that tag. It is up to the programmer to insure that there is room for the subroutine at the location specified. He may do this for example by appropriate use of control lines such as "RESERV" or "LOCATE", which are explained in detail later (Sections 14.2 and 14.4).

If the v address field of a line which produces a calling sequence contains a tag which does not appear in the tag field on any line of coding, or if it is blank, the subroutine will be placed in the compiled region and the tag, if any, will be equated to the machine locations, within the compiled region, associated with the first word of the subroutine.

The u address field of a line of coding which produces a subroutine calling sequence may contain any of the types of terms permitted in the address fields of a line which produces an 1105 instruction. It represents the location of the argument or parameter (or the first of these if more than one is required. If more than two parameters and/or arguments are required they must be available, in the correct order, in sequential cells so that they may be transferred to the correct cells within the subroutine by a repeated transmit positive instruction which will be produced by the compiler. If only two parameters or arguments are required they may be stored in sequential cells or in the left and right halves of the accumulator.

The compiler will insure that any subsidiary subroutines which are referred to by the primary subroutine are included in the compiled region unless the programmer, by means of the USES control line, which is explained in detail in Section 14.7, indicates that a copy of the subsidiary subroutine at some specified place outside of the compiled region is to be employed.

The calling sequence produced by the compiler will consist of either a single return jump; a transmit positive, return jump; a repeat transmit positive, return jump; or a left transmit, transmit positive, return jump, depending on the number of parameters and/or arguments required. In any case the tag, if any, in the tag field of the line which produces the calling sequence is equated to the machine locations associated with the first word of the calling sequence.

It is possible to include a subroutine in a program without producing a calling sequence. This is accomplished by means of the LOCATE control line which specifically locates, at the point in the program where the LOCATE line occurs, the subroutine whose name appears in the u address field.

It is, of course, the programmer's responsibility to insure that every subroutine to which he refers (and any subsidiary subroutines which they require) are available to the compiler at the time his program is compiled. Any subroutine, provided it is in the proper form, which is not in the internally stored library may be included with the coding for the main program. The compiler will accept such external subroutines, and in effect (though not physically) add them to the library while that one program is being compiled.

12. FORM OF SUBROUTINES

The compiler will compile subroutines which are in a magnetic tape internal library or included externally as part of the coding for the program. Both internal and external subroutines are written in the same form as other coding for the compiler with certain exceptions which are described here.

The item number field should be left blank on all lines of coding of a subroutine.

A subroutine consists, in general, of nine parts:

- 1) Leading line
- 2) Parameter lines
- 3) Entrance line
- 4) Alarm exit
- 5) Normal exit
- 6) Result cells
- 7) Argument and/or parameter cells
- 8) Body of the routine
- 9) Ending line

- 1) The *leading line* of a subroutine must have the special control symbol "SUB" in the operation code field to indicate that a subroutine is to be processed. The u address field of the leading line contains the name of the subroutine, and the v address field contains an integer which is equal to the number of consecutive cells occupied by routine. The "SUB" control line is described fully in Section 14.10.
- 2) There are three types of *parameter lines* introduced, respectively, by the special control symbols "TEMPS", "INOUT", and "Pn" (where n represents a digit 0 through 9) in the operation code field.

A "TEMPS" control line is used to indicate, by an integer in the u address field the number of consecutive cells used by the subroutine in the subroutine temporary pool in the compiled region. An integer in the v address field indicates the position within the temporary pool at which the temporaries for this subroutine are to start. Normally this is zero except when it is necessary to avoid conflict with a subsidiary subroutine which itself uses the temporary pool.

An "INOUT" control line is used to specify by means of integers in the u and v address fields, respectively, the number of arguments and/or parameters and the number of results. If either the "TEMPS" line or the "INOUT" line does not appear the corresponding parameters are taken to be zero.

Control lines introduced by "PO", "P1", . . . "P9 in the operation code field have no effect on the compilation of the program but are permitted to make USE subroutines compatible with other systems.

- 3) The *entrance line* produces the first word of the machine language subroutine. It should be an unconditional jump to the body of the subroutine.
- 4) The *alarm exit* line should have the control symbol "ALARM" in the operation code field. This produces the second word of the machine language subroutine. The compiler will supply an appropriate instruction for the cell. This will depend on the computation system in use at a particular installation and might, for example be a return jump to an alarm print out routine located at a fixed position on the drum.
- 5) The *normal exit* line produces the third word in the machine language subroutine. It should be an unconditional jump instruction which may be set up by a return jump instruction in the main program.

- 6) *Result cells* and
- 7) *Parameter and/or argument* cells must be reserved within the subroutine immediately following the normal exit either by use of the "RESERV" control line or by making entries in the tag field on the correct number of lines.
- 8) *The body of the subroutine* may contain the following types of lines:
 - Lines which produce 1105 instructions
 - Lines which produce numbers
 - Lines which produce subroutine calling sequences
 - "EQUALS" and "RESERV" control lines

These lines may be written just as in other (non-subroutine) coding for the compiler except that:

 - a. No constant pool addresses of the form "L(. . .)" may be used.
 - b. Tags which appear in the v address field of lines which produce subroutine calling sequences should not appear in the tag field. They are equated by the compiler to the machine locations associated with the first word of the subsidiary subroutine thus called for.
 - c. Other tags which appear in the u or v address fields but not in the tag field are equated, in order of their occurrence, to the addresses of cells in the subroutine temporary pool in the compiled region.
- 9) The *ending line* of a subroutine must contain the special control symbol "ENDSUB" in the operation code field.

13. CHANGES AND CORRECTIONS; RECOMPILING

The procedure described in this section in no way precludes any other method of correcting a program compiled by the USE Compiler. Since both a loadable binary tape and a side-by-side listing tape are produced as outputs from the compiler the installation or individual programmer has complete freedom to proceed in whatever way is most advisable in each case. The binary tape may be loaded into the computer and binary corrections inserted manually or via any of the input media, or a tape editing routing may be used to correct the binary tape.

However, when it is desired to obtain the advantages of a re-compilation (such as an up-to-date listing and freedom from patches) the compiler itself should be used. Since an output listing tape appears to the compiler to be the same, in all essential respects, as an original untyped main program tape, the compiler does not differentiate between the first and subsequent compilations of a program. Changes may be made at the time of the first compilation if desired.

Each program to be compiled may be accompanied by a list of changes on magnetic tape. The changes are written exactly like other lines of coding except that the item numbers on successive lines need not be in order. However, where a single digit or a blank appears in the item

number field of a change line the compiler will supply an item number just as it does for lines of coding in the main program. The changes are then sorted by the compiler according to the item number and merged with the lines of coding from the main program as compilation proceeds.

Three types of changes may be made: one-for-one replacements, insertions, and deletions. It is important to note that these are changes to the compiler language program, not to the machine language program. That is, lines of coding, not computer words, are being replaced, inserted and deleted. It is therefore entirely feasible to change control lines as well as lines which produce 1105 instructions or numbers. This is one of the advantages of recompiling.

Replacements: If the item number of a change line is identical to the item number on a line of the main program, the change line will replace the line in the main program.

Insertions: If the item number of a change line is not identical to the item number on any line of the main program the change line will be inserted between two lines in the main program at the point such that the resulting sequence of item numbers is monotonically increasing. This point is, of course, unique.

Deletions: If the first of a group of successive lines is replaced (as explained above) by a change line having the control symbol "DELETE" in the operation code field the entire group of lines will be deleted from the main program. The extent of the group of lines to be deleted is specified in the address fields of the "DELETE" control line as is explained in detail in Section 14.8.

14. THE USE OF CONTROL LINES

Many of the special control symbols which may appear in the operation code field have already been mentioned. In this section the uses of all such control lines are explained in detail.

14.1 "SETLOC" The "SETLOC" control line normally sets the execution and storage location counters to the translated values of the expressions found in the u and v address fields respectively. These fields may contain any of the types of terms permitted in the u and v address fields of lines which produce 1105 instructions (Sec. 6.1) with two exceptions:

- 1) Constant pool addresses of the form "L(. .)" are not allowed.
- 2) Any tags which are used must have appeared in the tag field on a preceding line of coding.

If either the u or the v address field is blank the corresponding location counter is not altered (thus a blank is not equivalent to zero in this case).

"SETLOC" is not allowed in subroutines. It should not have a tag in the tag field.

14.2 "RESERV" The "RESERV" control line normally advances the execution and storage location counters by the translated values of the expressions found in the u and v address fields respectively. These fields may contain the same types of terms as those described in the first paragraph of "SETLOC" in Section 14.1.

If the u or v address field is blank the corresponding location counter is not altered.

If a tag appears in the tag field of a "RESERV" line, it is equated to the values found in the

location counters before they are altered. Thus the tag is associated with the first cell in the reserved region.

Within a subroutine, but not otherwise, the translated values of the counters of the u and v address fields must be the same and the reserved region is loaded with binary zeros.

- 14.3 **"EQUALS"** The **"EQUALS"** control line which may appear anywhere in a program, equates the tag appearing in the tag field to the translated values of the expressions found in the u and v address fields. These fields may contain the same types of terms as those described in the first paragraph under **"SETLOC"**.

If either the u or v address field, but not both, is blank, both the execution location and the storage location equated to the tag will be the same, and equal to the translated value of the expression in the non-blank address field.

- 14.4 **"LOCATE"** The **"LOCATE"** control line causes the subroutine whose name appears in the u address field to be included in the machine language code at the point where the **"LOCATE"** occurs. The v address field is not used. No calling sequence is produced.

If a tag appears in the tag field it is equated to the cell associated with the first word of the subroutine (i.e. the entrance).

The subroutine must be available to the compiler at the time the **"LOCATE"** occurs, either in the internal library or by having been included in a preceding part of the coding.

"LOCATE" may not be used within a subroutine.

- 14.5 **"COMPAT"** The **"COMPAT"** (compile at) control line permits the programmer to override the automatic placing of the compiled region at the high numbered end of core storage, and/or the to equate a tag to the machine locations of the first cell of the compiled region (i.e. the beginning of the subroutine temporary pool). At most one **"COMPAT"** control line may appear in a program but it may appear anywhere except within a subroutine.

Normally the execution and storage locations of the compiled region are made equal to the translated values of the expressions found in the u and v address fields, respectively. These fields may contain any of the types of terms described in the first paragraph under **"SETLOC"**. If the u address field is blank the execution location of the compiled region will be at the high numbered end of magnetic core storage. If the v address field is blank the storage location will be the same as the execution location.

If a tag appears in the tag field of a **"COMPAT"** control line, it is equated to the machine locations of the first cell of the compiled region.

- 14.6 **"DUPx"** The **"DUPx"** (duplicate) control line reduces the amount of writing required when the entries in corresponding fields on each of a group of successive lines would be identical. If the group of lines is immediately preceded by the appropriate **"DUPx"** control line, then, in all the lines of the group except the first, those fields would be duplicates of corresponding fields in the first line may be left blank.

In writing the **"DUPx"** operation code the letters O, U, and V refer to the operation code field, and the U and V address fields respectively, and are used to indicate which of these fields are to be duplicated. Thus, the **"DUPx"** operation code symbol may take any of the following terms:

"DUPO" **"DUPU"** **"DUPV"** **"DUPOU"** **"DUPOV"** **"DUPUV"**

The special case "DUPOUV", where all three of the fields are to be duplicated, is discussed below.

The extent of the group of lines on which the duplication is to be effective is indicated as follows:

The u address field of the "DUPx" control line contains a single integer equal to the total number of lines in the group (including the first), and the v address field contains the word "TIMES".

The first line following the "DUPx" line must be written out in full in the normal fashion. The remaining lines of the group are written in the normal way except that the duplicated fields are left blank.

The compiler actually fills in the blank fields. Therefore, the "DUPx" line itself will not appear on the output listing but each line of the group will appear in its complete form.

The special case, "DUPOUV", where all three fields are to be duplicated is treated in a slightly different way. The first line of the "group of identical lines" is written out in full immediately following the "DUPOUV" line as usual. But, the remaining lines of the group are not written at all. The "DUPOUV" line and the following line will appear on the output listing. "DUPOUV" should not immediately precede "LOCATE" or "SUB" control lines.

"DUPx" may not be used within a subroutine.

14.7 "USES" As explained in Sections 11 and 14.4, the occurrence of a "LOCATE" control line, or of a line which generates a subroutine calling sequence indicates that a specified subroutine is to be included in the program. If this primary subroutine requires subsidiary subroutines the compiler, normally, will insure that such subsidiary subroutines are included in the compiled region and the primary subroutine will refer to them there. The programmer may, however, override this automatic inclusion of subsidiary subroutines by means of "USES" control lines following the line which specifies the primary subroutine, (i.e. either a "LOCATE" control line, or a line which generates the subroutine calling sequence).

The u address field of the "USES" control line contains the name of a subroutine which is subsidiary to the most recently referenced primary subroutine. The v address field of the "USES" control line must contain exactly one tag. The compiler will then assume that the subsidiary subroutine whose name is in the u address field of the "USES" control line is available at the location specified by the tag in the v address field, and the primary subroutine will refer to the subsidiary subroutine at that location. It is the *programmer's responsibility* to insure that the subsidiary subroutine is actually at the location indicated. This may be done, for example, by means of a "LOCATE" control line (See 14.4).

A "USES" control line must not appear within a subroutine.

14.8 "DELETE" The control line "DELETE", which should appear only among the changes, is used to delete a group of consecutive lines from the main program. The extent of the group of lines to be deleted is indicated as follows:

The u address of the "DELETE" line contains a single integer equal to the total number of lines in the group to be deleted (including the DELETE line itself) and the v address field contains either the word "LINE" or "LINES".

If both the u and v address fields of the "DELETE" line are blank, the effect is the same as though "1, LINE" had been written there.

The compiler actually carries out the deletion and neither the "DELETE" line itself nor the deleted lines will appear in the output listing.

14.9 "NOMORE" The "NOMORE" control line must appear only as the last line of coding in the list of changes and indicates to the compiler that there are no more changes. The content of the address fields is insignificant.

14.10 "SUB" The "SUB" control line may appear only as the leading line of a subroutine. The u address field contains the name of the subroutine. If the subroutine is in the internal machine library its name may consist of any combination of from one to six alphanumeric characters which does not conflict with any acceptable operation code symbol. If the subroutine is an external one supplied along with the main program its name must consist of one of the organization letter pairs such as those listed in Table 3 followed by from one to four alphanumeric characters.

The v address field of a "SUB" control line contains an integer which is equal to the number of consecutive cells required by the subroutine. This includes cells for the entrance, alarm exit, normal exit, results, arguments and instructions, constants, and subsidiary subroutine calling sequences in the body of the routine, but does not include cells used in the subroutine temporary pool.

14.11 "ALARM" The "ALARM" control line should appear only immediately following the entrance line within a subroutine. It will produce an appropriate alarm exit instruction in the machine language code. The contents of the address fields are insignificant.

14.12 "TEMPS" The "TEMPS" control line may appear at most once within each subroutine. It indicates to the compiler what part of the subroutine temporary pool in the compiled region is to be used by the subroutine in which it occurs. The u address field contains an integer which is equal to the number of consecutive cells used by this subroutine in the temporary pool. The v address field contains an integer which is equal to the number of cells to be left at the beginning of the temporary pool ahead of those used by this subroutine. These may be temporaries used by subsidiary subroutines. If either address field is blank, or if the "TEMPS" line does not appear within the subroutine the corresponding quantities are taken to be zero.

14.13 "INOUT" The "INOUT" control line may appear at most once within each subroutine. The u address field contains an integer which is equal to the number of parameters and/or arguments which must be placed within the subroutine before it is entered. The v address field contains an integer which is equal to the number of results which the subroutine computes and stores within itself. If either address field is blank, or if the "INOUT" line does not appear within a subroutine the corresponding quantities are taken to be zero.

14.14 "P_n" Control lines having "P₀", "P₁", . . . "P₉", in the operation code field are reproduced on the output listing but have no effect on the program. They are permitted in order to make it possible for USE subroutines to be compatible with other systems which might require more parameters. They may occur only within subroutines.

14.15 "ENDSUB" The "ENDSUB" control line may appear only as the final line of each subroutine.

The contents of the address fields are insignificant.

14.16 "END" The control line "END" must appear as the last line of the program. The content of the u address field is insignificant. The v address field may contain any of the types of terms permitted in the address fields of lines which produce 1105 instructions. The translated value of the entry in the v address field will be stored in a particular location in the final block on the binary output tape. It may be used by the binary tape loading routine as the address at which computation is to begin.

15. WARNINGS ISSUED BY THE COMPILER

During compilation of a program the compiler may detect various situations which make it impossible to continue in the normal way. These situations may represent typists' mistakes, peripheral or input equipment malfunction, programmer's blunders or intentional use of certain features of the compiler in an unanticipated manner. In practically all such cases the compiler will take a prescribed action to overcome the difficulty and then proceed with the compilation. A special warning symbol, "W", will be placed on the output listing on the line of coding in which the unusual situation was detected, and a footnote (referencing the item number of the line containing the warning symbol) will explain the reason for the warning. Table 4 lists some of the unusual situations which are detected and explains the action taken.

TABLE I
STANDARD OPERATION CODES

FP	Floating Polynomial Multiply	01	EJ	Equality Jump	43
FI	Floating Inner Product	02	QJ	Q-Jump	44
UP	Unpack	03	MJ	Manually Selective Jump	45
NP	Normalize Pack	04	SJ	Sign Jump	46
NE	Normalize Exit	05	ZJ	Zero Jump	47
TP	Transmit Positive	11	QT	Q-controlled Transmit	51
TM	Transmit Magnitude	12	QA	Q-controlled Add	52
TN	Transmit Negative	13	QS	Q-controlled Substitute	53
IP	Interpret	14	LA	Left Shift in A	54
TU	Transmit u-address	15	LQ	Left Shift in Q	55
TV	Transmit v-address	16	MS	Manually Selective Stop	56
EF	External Function	17	PS	Program Stop	57
RA	Replace Add	21	PR	Print	61
LT	Left Transmit	22	PU	Punch	63
RS	Replace Subtract	23	FA	Floating Add	64
CC	Controlled Complement	27	FS	Floating Subtract	65
SP	Split Positive Entry	31	FM	Floating Multiply	66
SA	Split Add	32	FD	Floating Divide	67
SN	Split Negative Entry	33	MP	Multiply	71
SS	Split Subtract	34	MA	Multiply Add	72
AT	Add and Transmit	35	DV	Divide	73
ST	Subtract and Transmit	36	SF	Scale Factor	74
RJ	Return Jump	37	RP	Repeat	75
IJ	Index Jump	41	ER	External Read	76
TJ	Threshold Jump	42	EW	External Write	77

TABLE II

j-TYPE OPERATION CODES

NEO } NEN }	Normalize	050	MS3	563
NE1 } NEQ }	Quasinormalize	051	PU0 } PU6 }	Punch 6 Levels 630
LTO } LTL }	Transmit A Left	220	PU1 } PU7 }	Punch 7 Levels 631
LT1 } LTR }	Transmit A Right	221	RPO } RPN }	Modify neither u or v 750
MJO		450	RP1 } RPV }	Modify v 751
Mj1		451	RP2 } RPU }	Modify u 752
MJ2		452		
MJ3		453	RP3 } RPB }	Modify both u and v 753
MJ4		454	ERO } ERA }	Read IOA 760
MJ5		455		
MJ6		456	ER1 } ERB }	Read IOB 761
MJ7		457	EWO } EWA }	Write to IOA 770
MSO		560		
MS1		561	EW1 } EWB }	Write to IOB 771
MS2		562		

TABLE III

PERMISSIBLE LEADING CHARACTERS FOR EXTERNAL SUBROUTINE

AP	Applied Physics Lab., Johns Hopkins University
BA	Boeing Airplane Company
CE	U. S. Army, Corps of Engineers
HO	Holloman Air Force Base
ML	Missiles Systems Division, Lockheed Aircraft Corp.
RR	Remington Rand Division, Sperry Rand Corp.
RW	Ramo-Wooldridge Corp.
WF	Wright Air Development Center
NC	University of North Carolina
AR	Armour Research Foundation

TABLE IV
WARNINGS ISSUED BY THE COMPILER

<i>WARNING</i>	<i>ACTION TAKEN</i>
More than 6 characters in tag or operation code field.	Rightmost six characters are used.
More than 59 characters in u and v address field.	Leftmost 59 characters are used.
More than 7 terms in u or v address fields.	Leftmost 7 terms are used.
Illegal term in u or v address fields.	Illegal term translated as zero.
Superfluous sign in u or v address fields.	Rightmost sign applied (i.e. sign closest to term).
Tag in u or v address fields which appeared more than once in tag field.	Location first equated to tag is used.
Constant pool full.	Address of form "L(. . .)" translated as zero.
Duplicate tag in tag field.	Location first equated to tag is used.
Tag incorrectly in tag field of control line.	Tag is ignored.
No room in directory for tag.	Tag will not be equated to a machine location.
Tag incorrectly used in u or v address fields of control line without having previously appeared in tag field.	Tag translated as zero.
Constant pool address of the form "L(. . .)" in u or v address field of control line.	Term translated as zero.
Translated value of expression in u or v address field $\geq 2^{15}$ or < 0	Used MOD 2^{15} .
Translated value of expression in u address field $\geq 2^{12}$ or < 0 in j-type operation.	Used MOD 2^{12} .
Illegal operation.	Translated to binary zero word.
Number contains too many digits, or an illegal character, or is too large or too small.	Number translated as binary zero.
No room in subroutine table, illegal name on external subroutine or subroutine not available to compiler.	Subroutine not included in program.
More than one "INOUT" or "TEMPS" control line in subroutine.	Last one to appear is used.
Too many temps, arguments, or results used by subroutine.	Subroutine not included in program.

TABLE IV (Continued)
WARNINGS ISSUED BY THE COMPILER

<i>WARNING</i>	<i>ACTION TAKEN</i>
Storage Regions overlap.	Will be loaded in order written with compiled region and subroutines last.
“DUPOUV” applied to certain lines illegally.	“DUPOUV” is ignored.
Improper entries in u and/or v address fields of “DELETE” control line.	Treated as though it read “1, LINE”.
More than one change with the same item number.	Last one used.
Both u and v address fields blank on an “EQUALS” control line.	Tag is equated to zero.
“USES” control line out of place.	Line ignored.
“COMPAT” used more than once in program.	First “COMPAT” applies.
Too many warnings.	Further warnings not footnoted.

TABLE V

XS3-n

<i>XS3 Character Desired</i>		<i>Write</i>
Digits 0 thru 9		Digits 0 thru 9
Letters A thru Z except 0		Letters A thru Z except 0
., -, +,), (, /		., -, +,), (, /
Space	(01)	*
Letter O	(51)	#0
Fast Feed 1	(37)	#1
Fast Feed 2	(42)	#2
Fast Feed 3	(57)	#3
Fast Feed 4	(76)	#4
Breakpoint β	(61)	#B
Comma ,	(21)	#C
Dollar sign \$	(55)	#D
Multiline	(20)	#M
Number sign #	(35)	#N
Stop code	(60)	#S
Ignore	(00)	#I

IV. Technical Notes on Operating the USE Compiler with the UNIVAC 1105

To compile a program:

The compiler must be stored in the computer. It occupies the first bank of core and 42100₈ - 47777₈ on the drum. The second bank of core is used during compilation, also cells 50000₈ - 67777₈ for storage of tags and their associated addresses and cells 70000₈ - 77777₈ for storage of changes. 42000₈ - 42077₈ are reserved for input and output parameters. Cells 40000₈ - 41777₈ are not used by the compiler. Note: Since part of the compiler loads directly into core and is modified during compilation, it is necessary to reload the compiler in order to restart or to compile a second program.

The following input parameters must be preset:

- 42000₈ - The compiler exits to this cell at conclusion of compilation. It should contain a stop order or a jump to an arbitrary program.
- 42001₈ - Any changes to the main program? 0 = no, 1 = yes
- 42002₈ - Type of main program. 0 = untyped, 1 = symbolic output (recompilation)
- 42003₈ - Is there a subroutine library tape? 0 = no, 1 = yes
- 42004₈ - Number of blocks to the beginning of the library in case tape is not positioned at the beginning of the library index (in v).
- 42005₈ - ALARM line - an exit to an arbitrary routine which the compiler will store in the ALARM line of subroutines.
- 42006₈ - Last address of core + 1 available to program being compiled (in u and v) - 00 20000 20000 for a 2-core machine.
- 42007₈ - Number of acceptable (i.e. RW, BC etc.) 2-letter manuscript subroutine headings (in u). This indicates the number of entries in the L4300 region. There are 10 at present and the parameter should read 00 00012 00000. Twenty-five more cells are available for additional entries and the parameter should be increased as entries are added.

All tapes necessary to the program being compiled must be positioned on the proper uniservos. The compiler assumes the following tape assignments which cannot be changed without altering the compiler coding:

TCU 1 No. 3	- Intermediate	TCU 2 No. 3	- Input (see discussion of multiple tape input)
TCU 1 No. 4	- Manuscript subroutines	TCU 2 No. 4	- Binary output
TCU 1 No. 5	- Binary subroutine library	TCU 2 No. 5	- Symbolic output
TCU 2 No. 2	- Input	TCU 2 No. 6	- Changes to the main program

An input tape must *always* be available on TCU 2 No. 2, as well as blank tapes on TCU 1 No. 3, TCU 2 No. 4 and TCU 2 No. 5. All other tapes are optional. The manuscript subroutine tape is a temporary tape used by the compiler only on those programs which include external subroutines (as opposed to those subroutines on the binary subroutine library tape)

Start at cell 7000₈. Compilation should proceed to a normal stop – a jump to 42000₈ with all tapes rewound – except for the following reasons:

1. If there are too many entries in the subroutine library index, an “i” will be printed on the flexowriter, compilation will halt and control will be transferred to cell 42000₈.
2. If there is a check sum failure in loading the subroutine library index, a “c” will be printed on the flexowriter, compilation will halt and control will be transferred to cell 42000₈.
3. In case of a tape read error which fails on all biases, backward or forward, a “p” will be printed on the flexowriter, the buffer will be cleared, the tape moved forward one block and the compiler will MS to the entrance of the re-read routine (cell 367₈ in the present version of the compiler.) It is possible to try reading this block again by starting at 367₈. In case there is any question about which tape failed, the external function read command is stored in cell 42053₈. The compiler does not try to determine whether the error was a parity or sprocket failure.

At the conclusion of compilation, certain output parameters are stored by the compiler as follows:

42054₈ – 42065₈ Block counts on TCU 1, tapes 1 thru 10 respectively
42066₈ – 42077₈ Block counts on TCU2, tapes 1 thru 10 respectively

Multiple tape input. If the compiler encounters a block of X's before reaching the END line, it takes this as a signal that there is further input on another uniservo. Specifically, if the tape on TCU 2 No. 2 ends with a block of X's, the compiler will expect further input on TCU 2 No. 3, and will rewind No. 2 with interlock. If the tape on unit 3 ends with a block of X's, the compiler will rewind No. 3 with interlock and switch back to unit 2, assuming that an operator has placed a new tape on this servo. The compiler will continue to switch back and forth between TCU 2 No. 2 and TCU 2 No. 3 until an END line is encountered.

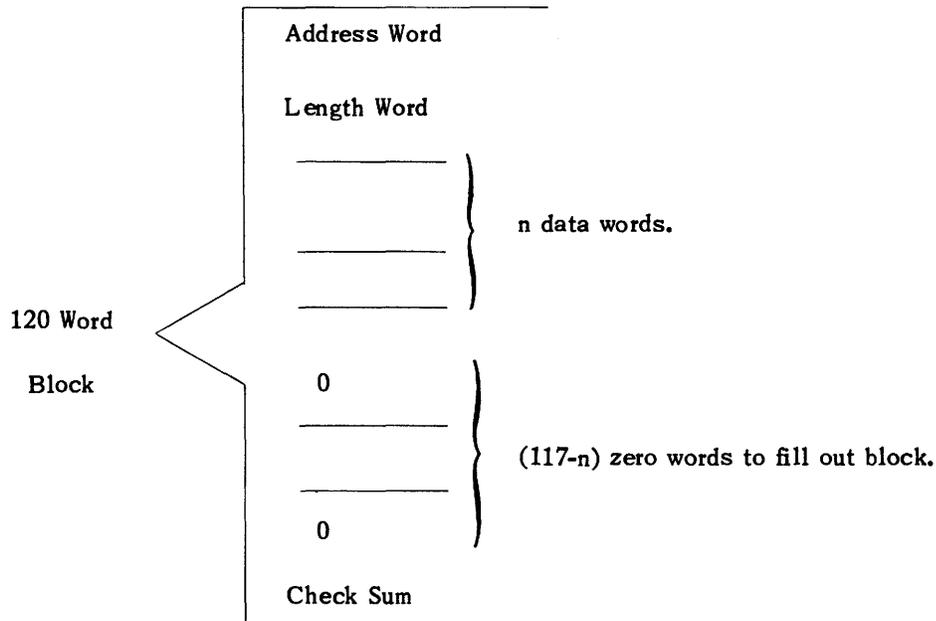
Extra block on input tapes. Since the input tapes to the compiler run free, it is necessary to write an extra block on each original input tape to avoid sprocket errors. It does not matter what is written in this block. The compiler will provide an extra block of space codes on the symbolic output tape for recompilation.

Midpoint stop. If the MS1 switch is set, compilation will halt before entering the second pass. By dumping the entire computer on tape at this point, it is possible to recover from a failure during the second pass. When re-starting, all tapes should be rewound except the binary subroutine library tape which *must* be positioned at the end of the index and before the first subroutine on the tape.

Binary output tape loader. The binary load routine is the same as that used on the 1103A. It is necessary to set the computer in the bypass mode from the console before using this routine.

FORM OF BINARY TAPE

Each block of the binary tape produced by the compiler has the following form:



An address word has the form:

```

00  EEEEE  SSSSS
6    15    15    bits
    
```

where EEEEE is the execution address, and SSSSS is the storage address of the first data word.

The length word takes the following form:

```

00  NNNNN  00000
    
```

The u address part of the length word, NNNNN, contains n, the number of data words in the block. [n ≤ 117 (decimal); usually n = 117.]

The length word in the last block takes the form:

```

40  00000  BBBB
    
```

where the v address part, BBBB, is the begin computing address (i.e. the translated value of the v field of the END line of coding). There are no data words in the last block.

The n data words immediately follow the length word. Following the data words are as many binary zero words as needed to fill out the block save one word.

The check sum, which is always the last word of the block, is right 36 bits of sum of the split extensions of all other words in the block. That is, the address word, the length word and the n data words are included in the check sum.

1105 USE COMPILER

ERROR CODES

000	No further errors
001	Too many characters in u, v, and comments
002	Superfluous signs in v
003	Illegal term in v
004	Too many terms in v
005	Next generated item number too big
006	Next item number illegal
007	This item number out of order
010	Duplicate tag in v
011	No room in directory for tag
014	L(c) in v of control command
015	Unassigned tag in v of control command
016	No room in directory for L (c)
020	$v \pmod{2^{15}} \neq v$
021	Illegal character in number
022	Too many digits in number
023	Number overflow
024	Exponent overflow
025	All significance lost
026	Illegal exponent
027	Duplicate tag in tag column
030	Item number of next change illegal
031	COMPAT repeated
032	Tag not significant with this control command
033	Improper u and v with DELETE
034	Change item number repeated
035	Too many characters in tag column
036	No letter in tag
037	Illegal character in tag
040	Compiled region too big
041	Two "USES" for same subsidiary subroutine
042	Too many subsidiary subroutines (table 6 filled)
043	More than 6n characters in XS3 instruction
044	Less than 6n characters in XS3 instruction
045	# followed by illegal character in XS3 instruction
134	ENDSUB inserted by compiler
135	This storage region overlaps another
136	Compiled region overlaps another region
137	Too many words in subroutine
140	Too many temporaries or arguments in subroutine
141	Extra INOUT or TEMPS in manuscript subroutine
142	USES operation out of place
143	LOCATE precedes SUB
144	Too many subroutine copies
145	Too many manuscript subroutines
146	Illegal duplication
147	Sub-copy omitted because tag directory is full

150 More than one sub with same tag
151 Illegal name on manuscript subroutine
152 u must equal v on this line
153 Line compiled at recycled location
154 $u \pmod{2^{15}} \neq u \pmod{2^{12}}$
155 Too many changes
156 Illegal operation
157 $u \pmod{2^{15}} \neq u$
160 Subroutine not available
161 No more room in table 7
162 Unassigned tag in u of control command
163 L (c) in u of control command
166 Too many overlapping regions
167 Duplicate tag in u
170 u and v blanks in EQUALS line
172 Error directory full
173 Too many terms in u
174 Illegal term in u
175 Superfluous sign in u
176 Too many characters in operation

1105 USE COMPILER

SUBROUTINE REGIONS

S0000	Sorting
S0100	Decode u and v
S0300	Tag directory routine
S0400	Line to symbolic output tape
S0500	Index location counters
S0600	Form 20-word symbolic blockette
S0700	Binary word to binary output tape
S0800	Get next line from intermediate tape
S1000	Op. switch on first pass
S1100	Number conversion (explicit)
S1300	Error routine
S1400	Evaluate u or v
S1500	Number conversion – L (c)
S1600	Process u or v
S1700	L (c) directory routine
S1900	Get next manuscript subroutine line
S2000	USES – first pass
S2100	B, F, or X – first pass
S2200	G or Y – first pass (NOT CODED)
S2300	2 octal digits – first pass
S2400	1105 or modified 1105 codes – first pass
S2500	SUB – interlude
S2600	SUB – first pass
S2700	DUPX – first pass
S2800	DUPOUV – first pass
S2900	COMPAT – first pass
S3000	EQUALS – first pass
S3100	RESERV – first pass
S3200	SETLOC – first pass
S3300	END – first pass
S3400	Illegal operation – first pass
S3700	LOCATE – first pass
S3800	Calling sequence generator – first pass
S3900	List only – first pass
S4000	List only – second pass
S4100	B, F, or X – second pass
S4300	2 octal digits, 1105 and modified 1105 operations – second pass
S4400	P(n), TEMPS, INOUT – interlude
S4700	Calling sequence generator – second pass
S4800	SUB, LOCATE, EQUALS, RESERV, COMPAT, SETLOC, USES, DUPOUV, ops. WITHIN SUBROUTINES – second pass
S4900	ALARM – interlude
S5000	RESERV – interlude
S5100	EQUALS – interlude
S5200	ENDSUB – interlude
S5300	1105 and modified 1105 operations – interlude

S5400 2 octal digits – interlude
S5500 END – second pass
S5600 Illegal operation – second pass
S5700 B, F, or X – interlude
S5800 Illegal operation – interlude
S6000 More subs. needed?
S6100 Check tag for duplication
S6300 List only – interlude
S6400 Calling sequence generator – interlude
S6500 Op. switch – interlude
S6600 Change bias and re-read
S6700 Changes to drum
S6800 Start of region
S6900 End of region
S7000 Tag assignment – first pass
S7100 SUB – postlude
S7200 P(n), TEMPS, INOUT – postlude
S7400 Set ups
S7500 Set up for first pass
S7600 Merge
S7800 2 octal digits, 1105 and modified 1105 operations – postlude
S8000 Set up for second pass
S8200 Refill the tape bin
S8300 Get next character
S8500 π (pi)
S8600 ρ (rho)
S8700 Line to intermediate tape
S8800 Set up for interlude
S8900 Line to manuscript subroutine tape
S9000 Set up for postlude
S9100 Subroutine op switch – first pass
S9300 Get next character from standard bin
S9500 B, X or F – postlude
S9700 Illegal operation – postlude
S9800 ALARM – postlude
S9900 RESERV – postlude
SA200 EQUALS – postlude
SA300 ENDSUB – postlude
SA400 List only – postlude
SA500 Calling sequence generator – postlude
SA600 Clean up following last ENDSUB
SA700 Search tables 2 and 3
SA800 Search table 1
SA900 Generate error list
SB200 Search table 7 (subroutine W)
SB700 Load subroutine library index
SC600 XS3 – first pass
SC700 XS3 – interlude
SC800 XS3 – second pass and postlude
SD200 Generate table 8
SD300 Pick up relocatable binary subroutines
SD400 Final clean up

1105 USE COMPILER

LIMITS

239 changes including the NOMORE line

4061 tags including A, Q, D, FILL and L

25 errors which will be indicated with codes as well as W's

120 subroutines in library index (including external routines)

50 subroutines in any one program (including subsidiaries)

100 subsidiary subroutines in library index (including external subs.)

25 subroutines referred to by USES commands

100 L (c)'s

50 storage regions where check for overlap will be made

25 overlapping regions where error will be indicated

63 arguments in a subroutine

1023 words in a subroutine

127 temporaries in a subroutine

63 results in a subroutine

1105 USE COMPILER

NOTES ON THE INTERMEDIATE TAPE FORMAT

A line of coding as it is written on the intermediate tape is 34 computer words long. They are arranged as follows:

- 1 Item number – integer in op and u, fraction in v
- 1 Alpha numeric tag
- 1 Op. jump (replacing the actual operation)
- 13 Tag, operation, u, v and comments
- 1 Location counters – in u and v
- 1 Code word for u – up to 7 five bit codes
- 7 Decoded u terms
- 1 Code word for v
- 7 Decoded v terms
- 1 Error indicator

34

1105 USE COMPILER

NOTES ON SUBROUTINE TABLES

TABLE 1 Col. 1 – L2200	Col. 2 – L2600	Col. 3 – L2900 (Subs. in program)
Name of subroutine Once for each copy needed	Location at which to assemble. Relative location of tag, blank or EAC and SAC. Op part indicates which.	<i>u</i> Relative location in tables 2 or 3. This gives implicitly the tape unit and the relative position on that tape. <i>v</i> Relative position (left 8 bits) and extent (right 7 bits) of associated entries in table 7 Set to 40 00000 00000 originally. Sign bit removed when col. 3 <i>u</i> is filled.

TABLE 2 Col. 1 – L1900	Col. 2 – L2000	Col. 3 – L2100 (sub. library index)
Name of subroutine	From left No. of temps. 7 bits Rel. pos. of temps 7 bits No. of words 10 bits No. of arguments 6 bits No. of results 6 bits	<i>u</i> – No. of cross references <i>v</i> – Location in table 5 of cross references

TABLE 3 Col. 1 – L1000 Col. 2 – L1200 Col. 3 – L1500 (ms. sub. index)

This is the same as table 2 except that column 3 refers to table 6 instead of table 5 and relative position matches manuscript subroutine tape instead of library tape.

TABLE 4 – L6300-A sorted list of table 1

A single list containing the relative positions of subroutines in table 1 in order of increasing magnitude of the third column. Repetitions are indicated by preceding zeros.

TABLE 5 – L3200 – Subroutine cross references

A series of groups of subroutine cross references for library subroutines. All those subroutines explicitly referred to in calling sequence generators within the subroutines of table 2 will be listed in the group associated with that subroutine.

TABLE 6 – L3300

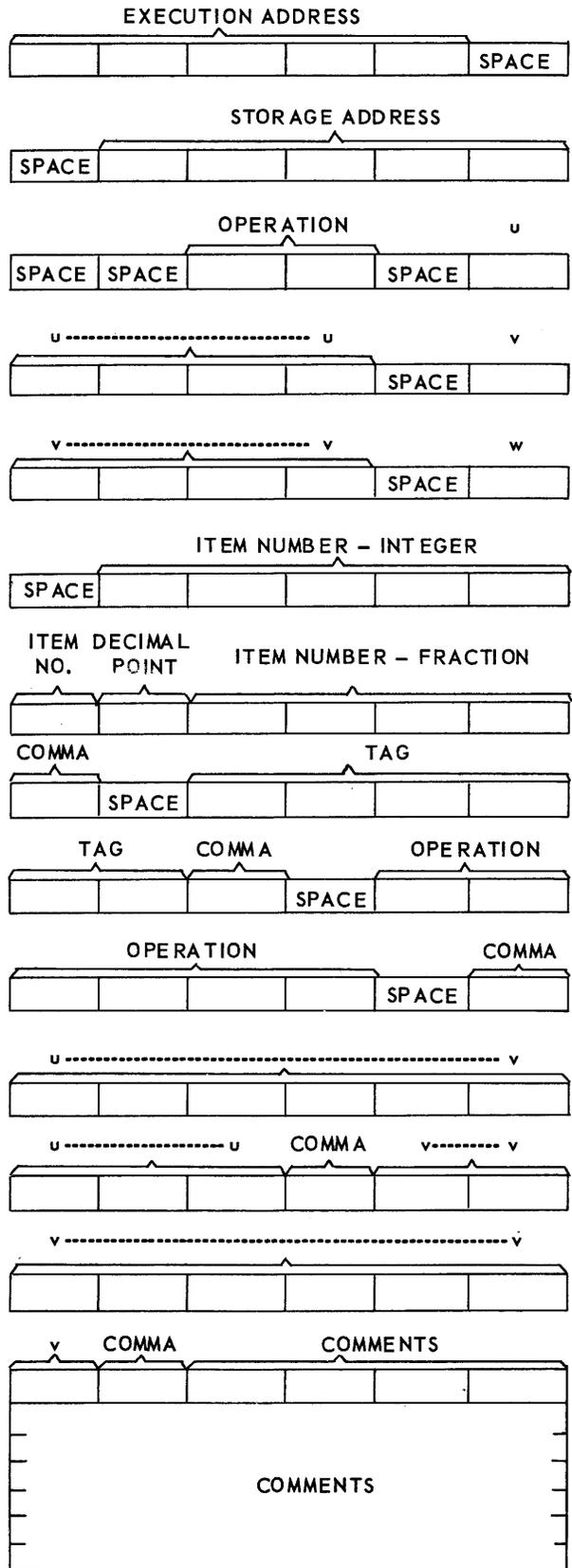
The same as table 5 except that it is associated with the manuscript subroutines in table 3 instead of the library subroutines in table 2. Table 6 may contain library or manuscript subroutines, while table 5 may contain only library subroutines.

TABLE 7 Col. 1-L3700	Col. 2 – L3800	Cross referenced subs mentioned in USES
Name of subroutine	Relative positions of tags from <i>v</i> of USES commands.	All references associated with a single copy of a given subroutine are grouped together in this table unless the programmer erroneously split up his USES commands

Note: The beginning of table 6 immediately follows the end of table 5, just as the beginning of table 3 immediately follows the end of table 2.

1105 USE COMPILER

FORMAT OF A BLOCKETTE ON THE SYMBOLIC OUTPUT TAPE



V. Examples Demonstrating Sundry Properties of the Compiler

1. Subroutines and calling sequences

a) Internal subroutine SQRT. Consider the line

```

39, JOE, SQRT, NUMBER, ROOT      $
40, FRED, TP, ROOT +3, A,        $

```

JOE is equated to the current values of the execution address counter and storage address counter just as it is for any ordinary line (e.g., both at 06000₈)

SQRT is searched for in the main subroutine library index when it is learned how many arguments and results cells are required. In this case there is 1 argument and the 1 result is set in the same cell so will be recorded as zero results.

NUMBER is the address where the first (in this case the only) argument is to be found (e.g. at 00407₈).

ROOT is equated to the entrance line of the subroutine. Suppose this is at 16013₈ onwards.

The calling sequence generated would be then: -

```

06000 06000 11 00407 16016 39. ,JOE, SQRT, NUMBER, ROOT, $
06001 06001 37 16015 16013
06002 06002 11 16016 32000 40. ,FRED, TP, ROOT +3, A, $

```

b) M.S. subroutine RRF004. This is also a square root routine (in fact the same as SQRT but in M.S. form). It contains all the information necessary to generate the same calling sequence as for SQRT when we write

```

,JOE ,RRF004, NUMBER, ROOT, $

```

Let us consider another example RWEG11. On the input tape will have been untyped the following -

```

, ,SUB, RWEG11, 461, $
, ,TEMPS, 1 , 0 , $
, ,INOUT, 2 , 2 , $

```

After this, comes the subroutine in standard form. In the program suppose we write –

```

99,   BOB, RWEG1, FRED, SID, $
100,  JOE, TP    , SID + 3, Q, $

```

Then reference to the INOUT line gives the following calling sequence generated by the compiler. (Assuming EAC = SAC at 00110₈, arguments at 04000 and 04001 and the entry line of the subroutine is subsequently set in the compiled region by the machine, say, at 16223₈)

```

00110 00110 75 30002 00112 99. , BOB, RWEG1, FRED, SID, $
00111 00111 11 04000 16230
00112 00112 37 16225 16223
00113 00113 11 16226 31000 100. , JOE, TP    , SID+3, Q, $

```

Notice that SID is assigned by the compiler in the compiled region. Another copy of the subroutine will also be found in the compiled region if another calling line for RWEG1 has its v field blank. In general this is undesirable.

2. SETLOC Consider the following:

```

71, FRED, RPB , 150, JOE, $
, , TP , JOE)S, JOE, $
, , SETLOC, 100, 50000)B, $
, JOE , TP , A , Q , $ etc.

```

This routine transfers 150 instructions from drum to core and jumps to the first instruction so transferred. Suppose at FRED the execution and storage address counters are both 05361₈. The SETLOC operation resets them to 00144₈ and 50000₈ respectively. The listing appears thus

```

05361 05361 75 30226 00144 71. ,FRED, RPB, 150, JOE, $
05362 05362 11 50000 00144 72. , , TP, JOE)S, JOE, $
, , SETLOC, 100, 50000)B, $
00144 50000 11 32000 31000 74. , JOE , TP, A , Q , $

```

If a tag appeared in the tag field of item 73. we would get e.g.

```

W 73. ,TAG, SETLOC \-----

```

and at the end of the listing we would find

73.0000 032 where 032 is the error code indicating that "TAG" has no significance.

3. RESERV Consider the following

```

84 ,JOE , , , 1, $
85 ,FRED, RESERV, 25, , $
86 ,SID , TP , A, FRED $
87 , , RA , SID, JOE, $

```

Suppose at item 84 the execution and storage address counters had been 02753₈ and 51106₈ respectively. The following listing will be produced.

```

02753 51106 00 00000 00001 84. ,JOE , , , 1, $
      85. ,FRED, RESERV, 25, , $
03005 51107 11 32000 02754 86. ,SID , TP , A, FRED, $
03006 51110 21 03005 02753 87. , , RA , SID, JOE, $

```

Notice that the contents of locations 02754 thru 03004 are not listed neither are they altered i.e. we may not assume them to contain zero, (unless this control line appears in a subroutine when the contents of the u and v addresses thereof must be the same).

4. EQUALS Consider the following

```

35, , MJ, 0, SID, $

```

and later

```

49, , MJ ,0 , JOE , $
50, SID, EQUALS, 600)B, , $
51, JOE, EQUALS, SID , , $

```

If at both items 35 and 49 EAC = SAC and are respectively 00300 and 00315, then the listing appears thus:

```

00300 00300 45 00000 00600 35. , , MJ , 0 , SID, $
and later
00315 00315 45 00000 00600 49. , , MJ , 0 , JOE, $
      50. , SID , EQUALS, 600)B, $
      51. , JOE , EQUALS, SID , $

```

Note that SID must have appeared in the tag field before it can be used in the u or v address of an equals line.

5. LOCATE

Let INTSUB be a non-self restoring subroutine on the library tape occupying 25 cells and having 3 arguments yielding 1 result. Consider the following:

```

47 , JOE , INTSUB, FRED , BOB , $
48 , , TP , BOB + 3 , A , $
49 , , RPB , 25 , SID , $
50 , , TP , TED)S , BOB , $
51 , , SETLOC, 1000)B , , $
52 , TED , LOCATE,INTSUB , , $
53 , , SETLOC, TED)S+25 , , $
54 , SID , EJ , 1600)B , 1500)B , $

```

Suppose EAC = SAC = 00200_g at item 47, the 3 arguments at 00500_g 1 and 2 and the subroutine to be stored and executed at 01000_g. The compiler will give the following listing:

```

00200 00200 75 30003 00202 47. , JOE , INSTUB , FRED , BOB, $
00201 00201 11 00500 01004
00202 00202 37 01002 01000
00203 00203 11 01003 32000 48. , , TP , BOB+3 , A , $
00204 00204 75 30031 00237 49. , , RPB , 25 , SID , $
00205 00205 11 00206 01000 50. , , TP , TED)S , BOB, $
51. , , SETLOC , 1000)B , , $
01000 00206 Facsimile of 52. , TED , LOCATE , INTSUB , , $
, , the subroutine
, , stored and
, , executed at
01030 00236 01000g
53. , , SETLOC , TED)S+25, , $
00237 00237 43 01600 01500 54. , SID , EJ , 1600)B , 1500)B, $

```

When the compiled program runs, the subroutine is entered and then restored for the next entry.

When a LOCATE line introduces a M.S. subroutine, that subroutine must have already been listed.

6. DUP x

Consider the following:

```

51, , TP , A , Q , $
52, , DUPOV , 4 , TIMES , $
53, , RA , JOE , 1000)B , $
54, , , JOE+1 , , $
55, , , JOE+4 , , $
56, , , JOE+8 , , $

```

Suppose EAC = SAC 00100_g at item 51, and that JOE is at 00300_g. The output listing will appear as follows:

```

00100 00100 11 32000 31000 51. , TP , A , Q , $
00101 00101 21 00300 01000 53. , RA , JOE , 1000)B , $
00102 00102 21 00301 01000 54. , RA , JOE +1 , 1000)B , $
00103 00103 21 00304 01000 55. , RA , JOE +4 , 1000)B , $
00104 00104 21 00310 01000 56. , RA , JOE+ 8 , 1000)B , $

```

Notice that item 52 has disappeared.

7. DUPOUV

Consider the following: -

```

19,      ,      MS1,      ,      FILL,      $
20,      ,DUPOUV,      4,      TIMES,      $
21,      JOE ,      B01, 01010, 10101,      $
22,      FRED ,      TP,      A,      JOE+1,,      $

```

Suppose at item 19, EAC = SAC = 00300g. The output listing will appear as follows:

```

00300 00300 56 10000 30000 19. ,      ,      MS1,      ,      FILL,      $
      ,      ,      DUPOUV,      4,      TIMES,      $
00301 00301 01 01010 10101 21. ,      JOE,      B01, 01010, 10101,      $
00302 00302 01 01010 10101
00303 00303 01 01010 10101
00304 00304 01 01010 10101
00305 00305 11 32000 00302 22. ,      FRED,      TP,      A,      JOE+1,      $

```

8. USES

Consider the line:

```

,      JOE ,      SQRT,      NUMBER,      ROOT,      $

```

This we know produces a calling sequence at JOE to enter the subroutine SQRT, at ROOT. Subsequently ROOT must be defined and space made for the subroutine. Suppose this is at 02000g. Now consider the following:

```

,      ,      ARCTAN,      X      ,      SID      ,      $
,      ,      USES      ,      SQRT      ,      ROOT,      $

```

The arctan subroutine will be stored at SID and at some point therein will be the RJ to SQRT subroutine. The USES line makes this RJ,ROOT + 2,ROOT.

9. DELETE

Consider the following:

```

05777 05777 15 05760 06003 38. ,      ,      TU ,      TED      ,      BOB ,      $
06000 06000 11 00407 16016 39. ,      JOE ,      SQRT,      NUMBER,      ROOT,      $
06001 06001 37 16015 16013
06002 06002 11 16016 32000 40. ,      FRED,      TP ,      ROOT+3,      A      ,      $
06003 06003 11 30000 31000 41. ,      BOB ,      TP ,      FILL      ,      Q      ,      $

```

Suppose the changes tape contains the line

```

39.      ,      ,      DELETE,      2,      LINES,      $

```

On recompilation, incorporating the changes we will get

```

05777 05777 15 05760 06000 38. ,      ,      TU,      TED ,      BOB,      $
06000 06000 11 30000 31000 41. ,      BOB,      TP,      FILL ,      Q      ,      $

```

Notice BOB is changed, "2 lines" refers to lines of symbolic and not the machine coding they produce. JOE & FRED will have dissappeared from the tag directory, the DELETE line does not appear in the output listing.

10. TEMPS

Suppose the compiled region starts at 16000g, i.e., this is the first location in the subroutine temporary pool. Let there be a subroutine containing the line,

```
      ,      , TEMPS, 3, 4,      $
```

Subsequently in the body of this subroutine suppose there is the line

```
      ,      ,      TP,  A, WS1,      $
```

Where WS1 is the first unassigned tag in the subroutine. Then this line will be translated as

```
11  32000  16004
```

ATED

A.

U-1869