RECOMP II USERS' PROGRAM NO. 1034

PROGRAM TITLE: SIGNAL CORPS RECOMP ALGEBRAIC TRANSLATOR - SALT

PROGRAM CLASSIFICATION: Executive and Control

AUTHOR: T. J. Tobias
U. S. Army Signal Engineering Agency
Arlington Hall Station
Arlington, Virginia

This program was modified and adapted for use with a subroutine package - RECOMP II Program No. 50 - P^3-2 by L. Raphael and A. W. England, and the text was rewritten by R. S. Lynn of Autonetics Industrial Products.

CHECKED BY: R. S. Lynn and H. D. Goddard

PURPOSE: The Signal Corps RECOMP Algebraic Translator (SALT) is a one pass compiler system which translates from algebraic statements into a SCRAP assembly language program. This program may then be assembled by use of the SCRAP processor.

DATE: January 1960

REISSUED: January 1961

Published by

RECOMP Users' Library

at

AUTONETICS INDUSTRIAL PRODUCTS

A DIVISION OF NORTH AMERICAN AVIATION, INC.
3400 E. 70th St., Long Beach 5, Calif.

TABLE OF CONTENTS

SUMMARY OF OPERATING INSTRUCTIONS

1. EXPRESSING PROBLEMS IN SALT FORMAT — Page 1
2. SALT STATEMENT GRAMMAR — Page 1 - 12
3. RESTRICTIONS — Page 13
4. OPERATING INSTRUCTIONS — Page 13 - 15
5. EXAMPLE — Page 15 - 26
6. CODING INFORMATION — Page 27

SUMMARY OF OPERATING INSTRUCTIONS

See §4 for Greater Detail

1. Load and verify SALT compiler
2. Press Start 1.
3. Type in SALT coding
4. After the last statement, END$, has been punched, tear off and label SALT-punched tape.
5. Load and verify SCRAP Assembly.
6. Place SALT-punched tape in photoreader.
7. Set sense switch B on if printout of first pass is desired. Set sense switch D on. Sense switch C must be off.
8. Press Start 1.
9. After END OF FIRST PASS is typed, advance several folds of blank tape and press Start 3. This punches SAVE material.
10. Tear off and label first pass punched tape, and place it in the photoreader.
11. Sense switch D is not used. Set sense switch C off. If printed copy of second pass, including absolute coding, is desired, set sense switch B on. Press Start 2.
12. When second pass has completed punching, tear off and label the tape. This is the SCRAP assembled program.
13. If the first and second passes were not run sequentially, or if memory contents were altered, load SCRAP Assembly, then the SAVE material. Place first pass punched tape in photoreader and go to step 11.
14. To use the SCRAP assembled program, load the program tape and PPP-2, or the single tape containing the assembled program plus subroutines from PPP-2. See §4.3 (h). Set the location counter to the origin, which is usually 3000.0, and press the Start button.

SIGNAL CORPS RECOMP ALGEBRAIC TRANSLATOR

SALT

INTRODUCTION

SALT is a RECOMP II program which allows the user to express his problem in a simple format, eliminating for the user the task of machine language programming and coding. SALT translates statements in this simple format to a RECOMP II symbolic code format which is later assembled by the SCRAP program into absolute machine language. The assembled program utilizes relevant subroutines contained in RECOMP II Program No. 50, Program Preparation Package Number Two.

This combination of programs enables the RECOMP II to offer what is known as "automatic programming."

1. EXPRESSING PROBLEMS IN SALT FORMAT

Expressing a problem in SALT language is much like writing in standard mathematical notation. Additionally, SALT language is used to provide for input and output of information. Statements in SALT language may be grouped as:

1.1 ARITHMETIC STATEMENTS - which are the algebraic equations used in solving the problem.

1.2 INPUT/OUTPUT STATEMENTS - which provide for input of information from the typewriter or photoreader and output of information on the typewriter, and

1.3 CONTROL STATEMENTS - which are used for transfer of control from one statement to another in the program. Such statements are used to control repetitions and conditional transfers.

2. SALT STATEMENT GRAMMAR

2.1 STATEMENT ELEMENTS

SALT statements consist of combinations of numbers, variables, subscripted variables, functions, expressions, and keywords.

2.1.1 NUMBERS

Numbers in SALT are the ordinary decimal numbers, in the common format. They are restricted in that they may contain no more than fifteen (15) characters, counting the decimal point, if used, as a character.

Neither the integral part nor the fractional part of a number may contain more than eleven (11) characters.

For example,

```
 723.
+7
-7
 +7.0
0.006
4.3953
-.3362
```

Plus signs may be omitted

2.1.2 VARIABLES

A variable is an all-alphabetic word containing at most eight (8) letters. See § 3.1 for restrictions on variable names.

Examples of permissible variables are:

```
DELTA          X
NSUBSIX        ITEM
```

2.1.3 SUBSCRIPTED VARIABLES

Subscripted variables have the form V(K) or V(K,J), where V is a variable and K and J are either numbers or variables. The names of a subscripted variable should contain at most seven (7) letters, because the assembly program, SCRAP, places a letter K at the beginning of the subscripted variable name. Using eight (8) letters may not cause an error in the machine language program, but the assembly printout will replace the first letter with a K, as mentioned above.

Subscripted variables in SALT may refer to elements in a one or a two dimensional list or table. Examples of proper subscripted variables are:

```
MATRIX(I,J)
MATRIX(ROW,COLUMN)
X(5)
X(J)
MATRIX(5,J)
VECTOR(P)
```

An example of a subscripted variable which is not permitted is TENSOR (I,J,K). If the user attempts to input a third dimension, the program will reject the input and result in an error return.

The subscripts themselves may represent mixed number values, but only the integer part is used in the machine language program. SCRAP will print out the fractional part on the assembly printout if a mixed number was entered in SALT. However, the machine language program shifts off the fraction part in computing addresses.

2.1.4 FUNCTIONS

Functions have the form F(E), where F is the alphabetic name of a function, and E is an expression as described under § 2.1.5. The following functions are defined in SALT and SCRAP.

SQRT(E)	means	$\sqrt[2]{E}$
SIN(E)		Sine of E (automatic angle reduction)*
COS(E)		cosine of E (automatic angle reduction)*
TAN(E)		tangent of E (automatic angle reduction)*
ARCTAN(E)		Arctangent of E
ARCSIN(E)		Arcsine of E
ARCCOS(E)		Arccosine of E
LOGTWO(E)		$\log_2 E$
LOG(E)		$\log_{10} E$ (common logarithm)
LN(E)		$\log_e E$ (natural logarithm)
ABS(E)		$\lvert E \rvert$
EXP(E)		e^E
EXPTWO(E)		2^E
EXPTEN(E)		10^E
ANGRED(E)		E modulo π, in radians

*Arguments of trigonometric functions must be in radians.

Additional functions may be defined by the user. See § 6.1.

2.1.5 EXPRESSIONS

An EXPRESSION is a statement element or a combination of elements. Numbers, variables, subscripted variables and functions are expressions.

If X and Y are expressions, then the following are also expressions:

+Y	X/Y	(means $X \div Y$)
-Y	X&Y	(means X times Y)
X+Y	X'Y	(means X^Y)
X-Y		
(Y)	((X)+(Y))	(means X+Y, illustrating use of more parentheses than are required)
(X)		
(-Y)		
(+Y)		

Note that using parentheses when not required is permitted. If the user attempts to input X++Y or any other case of two signs in juxtaposition, the program will reject the statement and error return. Examples of expressions are:

A&(X'2)+B&X+C meaning AX^2+BX+C

(A+Z)/(N-1) - SQRT(2&A) meaning $\frac{A+Z}{N-1} - \sqrt{2A}$

(X(I,J)/K) & ARCTAN(1.770) - Y'LOG(K/J) meaning $\frac{X_{ij}}{K}$ Arctan 1.770 - $Y^{\log_{10}\frac{K}{J}}$

2.1.6 KEYWORDS

Keywords relate elements in SALT statements in a manner that is easier to illustrate than to define. In the following discussion, a "tag" refers to a symbolic location assigned to a SALT statement. See § 2.1.6.3.

2.1.6.1 ARITHMETIC STATEMENTS

Keywords used in arithmetic statements are:

:	equality sign
+	addition
-	subtraction
&	multiplication
/	division
'	exponentiation
$	terminates the statement

In general, arithmetic statements have the form:

Variable: expression $

The value of the variable is defined by the expression. For example:

```
LIST(A,B):SQRT(LIST(1,19))+10&ALPHA$
GROUP:GROUP+1$
```

Arithmetic statements may have a location tag. In the following examples, MODIFY and COMPUTE are location tags assigned to their respective statements.

```
MODIFY, X:X-1$
COMPUTE, ELEMENT(5):ELEMENT(5)/ELEMENT(J)$
```

An arithmetic statement may contain any EXPRESSION that is proper under the conditions of § 2.1.5, describing EXPRESSIONS.

2.1.6.2 INPUT/OUTPUT STATEMENTS

Keywords used for input/output are:

READY, READZ, PRINT, CXP, TAB, and CRR

READY statements have the form

```
READY  variable$ or
READY  subscripted variable$
```

READY statements may have a location tag. One value only is typed in on a READY statement. Examples of proper READY statements are:

```
01, READY  X$
READY  Y$
INPUT,  READY  MATRIX(I,J)$
READY  TABLE(ITEM,6)$
23,  READY  DATA$
```

The format of the value typed in is discussed fully in the description for Program Preparation Package Number Two.

These examples of proper formats should make the input format clear. The format is variable, as shown.

```
+1        7.53
-1       -4
 1       +3.9599
+1.0     698.7634
```

The numbers are typed using the top keys of the typewriter using any common format for the values. In addition, extremely large or small values may be input using the notation recognized by AN-007.1. 21+16 means 21 X 10^{16}, 14-10 means 14 X 10^{-10}. 1.6+6 means 1.6 X 10^{6}, and so on. The form may be expressed $\pm a \pm b$, where "a" is any proper value in common format times 10 to the $\pm b$ power, where $|a| \leq 2^{39}-1$, and $|b| \leq 511$.

Terminate the typing of the value with a carriage return, tab, space or blank.

READZ statements have the form

```
READZ  variable$ or
READZ  subscripted variable$
```

READZ statements will not carry location tags thru the second pass of SCRAP. Therefore, they may not be addressed by any tag given them in SALT. If it is necessary to address them, a dummy statement before the READZ statement may be used. The dummy statement is of the form

```
tag, CONTINUE$ or
tag, CRR$
```

The READZ causes the photoreader to try to read paper tape and to store the input information in the memory, starting at the location assigned to the variable. The information may be in any proper mode - N, C, F, or L. The tape may be prepared off-line. After the last word to be entered, there must be at least eight (8) spaces, then L22131 C/R S. The eight spaces allow the logic time to change modes, the L22131 sets the location counter to location 2213.1, the carriage return acts as an enter code, and the letter S acts as a start code.

PRINT statements have the form

```
PRINT  variable$
```

For example,

```
PRINT  ANSWER$ or
PRINT  MATRIX(K,J)$
```

Print statements may have a location tag. The PRINT statement types the value of the variable in fixed point format if the exponent -- power of ten -- of the variable is between -13 and +39. Otherwise, the value is typed in floating point format. Any formatting of output must be taken care of by the user, as no carriage return nor tab is included in the PRINT subroutine.

CXP statements have the form

```
CXP$
```

The CXP statement may be used as a check-point during program debugging. If Sense Switch B is on, it will type the absolute memory location of the CXP command that will be in the final SCRAP assembled machine language program, followed by the contents of the A and R registers in floating point format. The A and R are not destroyed. CXP statements may carry a location tag.

TAB statements have the form

```
TAB$
```

TAB statements may be tagged. This command causes the typewriter to tab

if and only if the tab defeat switch is in the off position. Otherwise, the typewriter will carriage return.

CRR statements have the form

```
CRR$
```

CRR statements may be tagged. This command causes the typewriter to letter shift and carriage return.

2.1.6.3 CONTROL STATEMENTS

A location tag is the symbolic name of a certain location. Location tags are involved in most control statements. Tags are either all Arabic numerals or all alphabetic characters. The numerals may be from 00 through 99. The location 00 is distinct from 0. If more than two digits are attempted as a tag, the SCRAP assembly will cut off and ignore the excess leading numerals. As many as eight alphabetic characters may be used for a tag. Array and routine names should be restricted to seven letters, because the SCRAP program places a K or an R in front of the name, and if there are already eight letters, a misspelling will occur. This does not necessarily cause an error in the final machine language problem, however. A tag identifying a statement is followed by a comma, as in the examples

```
DONE,GO TO FINISH$
18,K:K+M$
```

GO TO statements are of the form

```
GOTO tag$ or
GO TO tag$
```

The GOTO statement may be tagged. This statement unconditionally transfers control to the location specified by the tag in the address. Examples of GOTO statements are

```
23, GOTO START$
GOTO 14$
ENDPASS, GO TO ITERATE$
```

IF statements have two forms;

(a) IF (expression) minus, zero, plus $
(b) IF (SENSE n) on, off $

These statements may be tagged. In form (a), three location tags follow the parentheses. They need not be distinct; that is, two tags may be the same, or all three for that matter. Control is transferred to one of these locations according to the value of (expression).

In form (b), n stands for B, C, or D, referring to the three sense switches on the console. Two location tags follow the parentheses. Control is transferred to one of these locations according to the condition of the

sense switch referred to. Examples of proper IF statements are

```
TEST,IF(X-3.4) REDO,REDO,OUT$
IF(SENSE D) OUTPUT,ITERATE$
```

In the first example, if X-3.4 is negative or zero, control is transferred to location REDO. If X-3.4 is positive, control is transferred to OUT. In the second example, if sense switch D is on, control is transferred to OUTPUT. Otherwise, control is transferred to ITERATE.

DO statements have the form

```
DO tag FOR variable l.l.(inc.)u.l.$
```

The DO statement controls iteration loops. The variable may be a variable or a subscript. The value of the variable goes from l.l., lower limit, to u.l, upper limit, by increments of inc. The tag specifies the last statement in the range of the DO statement. For example,

LOOP, DO SWITCH FOR K 1(1)M$ means "perform all statements starting just past the DO statement down to and including the SWITCH statement, with K=1 the first time, K=2 the second time, and so on until incrementing K by 1 would make it greater than M."

```
DO ITERATE FOR K 1(2)7$
FUNCTION: THETA(K)/SIGMA(K)$
CRR$
PRINT FUNCTION$
TAB$
ITERATE, PRINT K$
```

This DO loop would print out

(actual value of function)

θ_1/σ_1	1.0000000000
θ_3/σ_3	3.0000000000
θ_5/σ_5	5.0000000000
θ_7/σ_7	7.0000000000

Do loops may be contained within the range of another DO loop. A simple example is

```
DO OUTPUT FOR K 1(1)3$
DO OUTPUT FOR J 1(1)4$
PRINT MATRIX(K,J)$
OUTPUT,CRR$
```

Range of inner loop

Range of outer loop

This would print

M_{11}

M_{12}

M_{13}

M_{14}

M_{21}

M_{22}

M_{23}

.

.

.

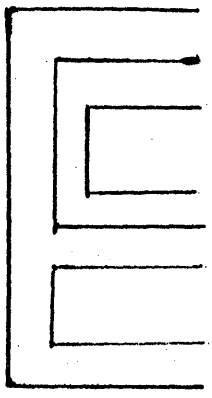
M_{34}.

Note that the inner DO loop is done before control is transferred to the outer DO loop. The DO loops are said to be "nested."

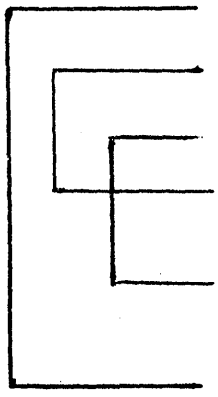
Note some restrictions on the use of DO loops. In the diagrams, the brackets represent the range of statements under control of a DO statement.

(a) If the range of a DO statement includes another DO statement, all statements in the range of this second statement must also be in the range of the first DO statement.

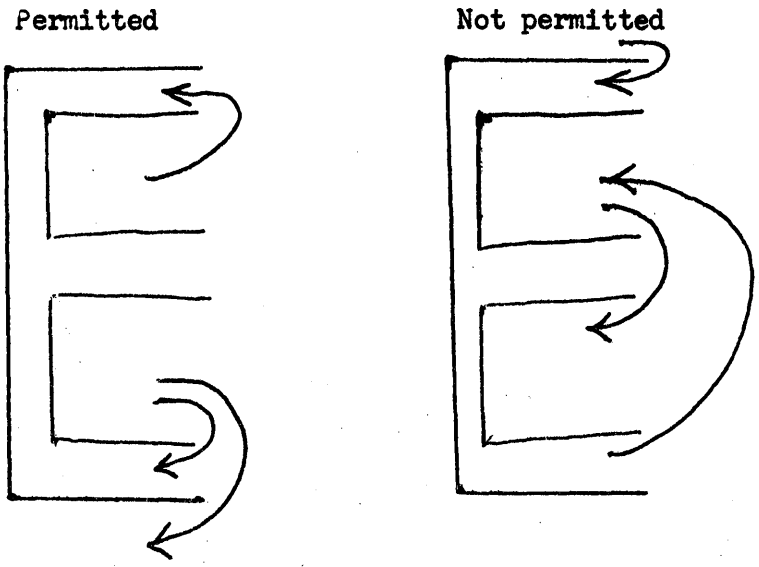
Permitted



Not Permitted



(b) No transfer of control by IF or GOTO statements is permitted into the range of any DO statement from outside its range, since such transfers would not permit the DO loop to be properly indexed.



(c) The last statement in the range of a DO loop may not be an IF or a GOTO statement. A CONTINUE statement may be used as a dummy last statement.

STOP statements have the form

```
STOP$
```

A STOP statement may be tagged. A halt is generated in the final machine language program. The STOP should be used only at the end of a program, usually, because it is not possible to continue the program after a STOP merely by pressing start button. The location counter would have to be changed before the program could be continued.

CONTINUE statements usually have the form

```
tag, CONTINUE$
```

CONTINUE statements are dummy statements which are used as last statements in a DO loop instead of an IF or GOTO, and as a means of tagging an ARRAY or READZ, which may not be directly tagged. For example:

```
START, CONTINUE$
ARRAY REGION(5,10)$
RDZ, CONTINUE$
READZ DATA$
```

ARRAY statements have the form

```
    ARRAY subscripted variable (items)$
or  ARRAY subscripted variable (rows, columns)$
```

ARRAY statements may not be directly tagged, but may be addressed using a dummy statement, as described in the preceding paragraph. ARRAY statements save storage space for vectors and matrices, which are to be defined and used in the program. ARRAY statements must come before any reference whatsoever to a subscripted variable, and it is provident to make all ARRAY statements the first statements in a SALT program.

ARRAY LIST (23)$ would save storage for 23 floating point values in a region called LIST. ARRAY TABLE (5,6)$ would save storage for 5 X 6 = 30 floating point values in a region called TABLE.

PAUSE statements have the unique form

PAUSE$

The SCRAP assembly will halt at this point in the program. For use of this special instruction, see the SCRAP program description.

ENTER SCRAP statements have the unique form

ENTER SCRAP$

This permits the typing of SCRAP coding in the manner described in the SCRAP program description. One enters SCRAP coding to accomplish that which may not be done with SALT statements. The SALT compiler ordinarily generates SCRAP coding for assembly by the SCRAP program. There are no provisions in SALT for generating alphanumeric output. This problem is solved by entering the proper SCRAP coding directly, using the ENTER SCRAP statement.

To input SCRAP coding, set margins at 10 and 90, and tab stops at 20, 35, and 50. Suppose that two function values had been computed using regular SALT coding; say X and X prime. Let these values be in symbolic locations X and XPRIME. The SALT and SCRAP coding to label and output these values could be:

```
CRR$

ENTER SCRAP$

  tab          CLA     tab    A     tab    L/S X F/S:       C/R
  tab          TYA     tab    N     tab    7760             C/R
  tab          FCA     tab          tab    X                C/R
  tab          PRINT   tab          tab                     C/R
  tab          CRR     tab          tab                     C/R
  tab          CLA     tab    A     tab    L/S X F/S ' :    C/R
  tab          TYA     tab    N     tab    7760             C/R
  tab          FCA     tab          tab    XPRIME           C/R
  tab          PRINT   tab          tab                     C/R
  tab          HALT    tab          tab                     C/R
GO TO SALT     tab     tab          tab                     C/R
```

In the above coding, L/S means letter shift, F/S means figure shift, and C/R means carriage return. The address 7760 is a special code used to type alphanumeric characters. No more than 8 characters including shifts may be in the heading. A and N are codes to identify the information that follows the tab. "Tab" means depress the tab key. The mnemonic codes CLA, TYA, etc., are typed in as shown. Depressing the carriage return key terminates each line of SCRAP coding.

Always return to SALT to make an END$ statement. SALT statements are accepted after the line of SCRAP coding:

```
GOTOSALT    tab   tab   tab   C/R
```

ROUTINE statements have the unique form

```
ROUTINE name$
```

where "name" is the name of the subroutine. This statement allows the construction of very simple subroutines. The use is illustrated in the description of the RETURN statement.

RETURN statements have the form

```
RETURN name$
```

where "name" is the name of the subroutine. The RETURN statement may be tagged with the permissible numerals or up to 7 alpha characters. Examples of ROUTINE and RETURN statements are

```
ROUTINE FOFX$
Y:(BETA & X'3 +(BETA -7) & X'2)'(BETA-2)$
RETURN FOFX$
```

or

```
ROUTINE FACTOR$
      .
      . (subroutine)
      .
GOBACK, RETURN FACTOR$
```

When a subroutine is to be used, it is entered with a GOTO name statement or an IF statement. The SALT statements defining a subroutine should be near or at the end of the SALT program, after a STOP or GOTO statement which would prevent control passing sequentially to the coding in ROUTINE. In other words, do not place ROUTINE definition in the middle of a program, unless care is taken to transfer control around the routine.

END statements have the unique form

```
END$
```

This must be the very last statement in every SALT program. The compiler will not accept further statements.

3. RESTRICTIONS

3.1 The following letter combinations may not be used as names of variables or subscripts.

(a) the single letter C
(b) any function name defined in SALT or SCRAP
(c) any symbolic command or pseudo-operation code defined in SALT or SCRAP, such as CLA or HALT.
(d) the name of any array prefixed by a K or an R. If, for example, VECTOR is a subscripted variable, do not use RVECTOR or KVECTOR.
(e) the name of any subroutine prefixed by an R. If ROUTINE SORT is defined in the program, do not use RSORT.
(f) any location tag

3.2 Certain restrictions must be recognized in the use of tags.

(a) Tags must be all alpha characters -- up to eight letters -- or all Arabic numerals -- from 00 to 99. Note that 00 is a different tag from 0, as 01 is different from 1.
(b) Do not use tags on a READZ statement or an ARRAY statement.

3.3 No statement containing an arithmetic expression may contain more than 128 variables, numbers, and keywords.

4. OPERATING INSTRUCTIONS

4.1.1 Load SALT tape. The memory will be cleared by a short program at the beginning of the tape, after which the tape will continue to load.

4.1.2 The tape may be verified by placing the tape in the photoreader past the short memory zero program and pressing the Verify button.

4.1.3 Set typewriter margins at 10 and 90. Set tab stops at 20, 35, and 50. Set tab override switch in the off position.

4.1.4 Advance at least two folds of blank tape.

4.1.5 Press Start 1 to begin.

4.2 TYPING STATEMENTS

4.2.1 The ALPHA light must be on and the COMPUTE light off before typing each character.

4.2.2 Each number or name must be separated by a space, figure shift, letter shift, or carriage return. A tab does not separate.

4.2.3 The symbols + - & / ( ) ' : $ and comma are individually recognized if they are the first symbols of a new field, but not when contained within a field.

4.2.4 Typing extra spaces, letter shifts, figure shifts, or carriage returns is permitted, where they do not change the total meaning.

4.2.5 Examine each statement for errors. If there are none, type $ symbol to enter the statement. If errors are discovered, type "line feed" symbol next to the M key, and type in the correct statement. If an error is recognized after the $ has been typed, and before the tape has begun to punch, it may be corrected by depressing the stop button, error reset button, and Start 1. Enter the correct statement. If the incorrect statement has started punching, there is no remedy but to start the whole operation over again, back to § 4.1.1. It is possible sometimes to patch the error by SALT statements which will nullify the error, or to correct the error during the SCRAP first pass, but in general, this is difficult and perhaps unrewarding. If the compiler program finds format errors in the statement, there will be an ERROR printout. Discover source of error, usually one of field termination or parenthesis usage, and retype the correct statement.

4.2.6 If there is an output error due to fast typing, press Error Reset button, Start 1, and retype the statement.

4.2.7 If it is desired to enter SCRAP coding, the SCRAP program description should be consulted. For simple typing of labels, see remarks under ENTER SCRAP statement.

4.2.8 TERMINATION - Every SALT program must end with the statement END$.

4.3 Using SCRAP assembly program

It is not necessary to be familiar with SCRAP to be able to process most SALT programs. The procedure for using SALT and SCRAP is essentially as follows:

(a) Load SALT compiler tape, and type in the SALT statements making up your program. After the last statement has been punched, advance several folds of leader and tear off the tape, leaving at least 2 folds for the next tape. It is provident to write "SALT output, _____ program" on the tape punched by the SALT compiler program. Reset the margins and tabs if they have been changed.

(b) Load the SCRAP Assembly tape. Set sense switch C off. Set sense switch B on if printout of the first pass of SCRAP is desired. The assembly is very much faster if B is off. Turn sense switch D on. This will make absolute assignment of symbolic locations on the first pass.

(c) Place the SALT punched output tape in the photoreader and press Start 1.

(d) After the typewriter prints END OF FIRST PASS, advance several folds of blank tape, and press Start 3. This is the SAVE routine of SCRAP. This is necessary in order to perform the second pass of the assembly in the event the computer memory contents will be changed between passes.

It is improvident to ignore this procedure. After the SAVE material is punched, advance several folds of leader and tear off the first pass output. Leave at least 2 folds of tape for the next output leader. Label this tape "SCRAP FIRST PASS, _____ program" to avoid confusing it with the 5 other tapes which will be around the computer.

(e) Sense switch C must be off. Place the SCRAP FIRST PASS tape in the photoreader and press Start 2. If there has been any change in the contents of computer memory since the first pass was punched, reload the SCRAP Assembly program, then the SAVE tape, then place the SCRAP FIRST PASS tape in the photoreader and press Start 2.

(f) When the second pass is completed, the absolute machine language tape has been punched. Tear off and label this tape with the name of the program. This tape will be much shorter than either the SALT output or the SCRAP FIRST PASS tape. This final tape will be in command format if sense switch B was on, and in the more compact alpha format if sense switch B was off.

(g) The final program is not yet in memory. If it is desired to try the program, it must be read in through the photoreader. Then it may be redumped with any dump routine if desired. After loading the final program, load PPP-2, RECOMP II Program No. 50. This will halt at 3000, the starting point of the final program. Press Start to begin.

(h) If it is desired to have the entire program on a single tape, see the description of PPP-2 for the procedure of punching those subroutines utilized. When this is done, be certain to punch also the calling sequence regions.

5. EXAMPLE

An example of the use of SALT and SCRAP coding to solve a Mortgage Amortization problem is shown. The first and second passes of the assembly are included.

It is desired to compute the monthly payment on a loan, and to obtain a complete amortization schedule including payment number, principal, interest, and new balance. The formulas to compute the desired results may be found in standard textbooks on the subject.

The relationships may be expressed as:

Monthly rate (MONRATE)=yearly rate $\div$ 12.

Monthly payment (MONPAY)=balance x monthly rate $\div$ (1-(monthly rate +1)$^{-N}$)
where N is the number of monthly payments.

Interest (INTEREST)=balance x monthly rate.

Principal (PRINCIPL)=monthly payment - interest.

Balance (BALANCE)=balance - principal.

The monthly payment will remain constant, the interest and balance for each month become less, and the principal becomes greater.

In SALT coding, these relationships may be expressed as:

```
MONRATE:RATE/12$

MONPAY:(BALANCE&MONRATE)/(1-MONRATE+1)'N)$

INTEREST:BALANCE&MONRATE$

PRINCIPL:MONPAY-INTEREST$

BALANCE:BALANCE-PRINCIPL$
```

These arithmetic statements are used in the SALT coding in the order shown. MONRATE and MONPAY are computed only once. The INTEREST, PRINCIPL, and BALANCE are computed for each month, under the control of a DO statement. The DO statement contains the lower and upper limits of the payment number, and the amount by which it is incremented.

SALT language may not be used to output headings. The more detailed SCRAP language must be used for this purpose.

Explanation of Mortgage Amortization Program Coding

The tag START is used at the beginning to permit repetition of the program with new data. SCRAP must be entered to type alphanumeric information. In the line of coding tagged AREF, the letter shift (L/S) might have been omitted, because after the initial carriage return, the typewriter is left in letter shift. In line BREF, the spaces (sp) are for appearance. Similarly, other spaces in alphabetic type addresses are used to position the output. Blanks (b) are typed to make a total of 8 characters. The code 777n or 776n is used to specify the number of characters to be typed. An n of zero specifies 8 characters. From 1 thru 7 characters are specified by an n of 1 thru 7 respectively.

After SCRAP is entered the second time, note that no L/S is used before MONTHLY because the CRR leaves the typewriter in letter shift. The CRR macro types a L/S C/R.

The 3 READY commands are for the purpose of entering, from the typewriter, the balance, rate and N when using the final assembled program.

The command pair

```
CLA     C     +0006020+5705300
STO     N     2256
```

modifies the output subroutine calling sequence to allow 6 places to the left of the decimal point, and 2 places to the right.

The DO statement specifies the first payment number as 1, and increments this number by 1 each time the statements in the loop are performed, until

the computations for the N^{th} payment are performed. The program then transfers to START, and the program may be repeated with new information if desired.

In the Assignment Table of the first pass printout, the alpha characters designated by ALFCN05, ALFCN06, ALFCN10, and ALFCN12 are printed in figure shift rather than letter shift. This is not an error. The typewriter is left in figure shift after typing the figures (XX) of any ALFCNXX. The alpha characters of the other ALFCN words are in letter shift because a letter shift was typed at the beginning of each of the address fields.

In the sample problem, 1200 was entered after BALANCE $ was typed, .09 was entered after RATE was typed, and 24 was entered after N was typed.

```
START,CRR$

ENTER SCRAP$
AREF        CLA         A   L/S BALANCE
            TYA         N   7770
BREF        CLA         A   F/S sp sp $ bbbb
            TYA         N   7764
            READY
            FST             BALANCE
            CLA         A   L/S RATE sp sp
            TYA         N   7777
            READY
            FST             RATE
            CLA         A   L/S N sp sp bbbb
            TYA         N   7774
            READY
            FST             N
GOTOSALT

MONRATE:RATE/12$

MONPAY:(BALANCE&MONRATE)/(1-(MONRATE+1)'(-N))$

ENTER SCRAP$
            CRR
            CLA         A   MONTHLY sp
            TYA         N   7770
            CLA         A   PAYMENT b
            TYA         N   7777
            CLA         C   +0006020+5705300
            STO         N   2256
            FCA             MONPAY
            PRINT
            CRR
            CRR
            CLA         A   sp sp sp sp sp bbb
            TYA         N   7775
            CLA         A   PAYMENT
            TYA         N   7770
            TAB
            CLA         A   L/S sp sp sp sp sp PR
            TYA         N   7770
            CLA         A   INCIPAL
            TYA         N   7777
            TAB
            CLA         A   L/S sp sp sp sp INT
            TYA         N   7770
            CLA         A   EREST bbb
            TYA         N   7775
```

```
          TAB
          CLA       A   sp sp sp sp sp bbb
          TYA       N     7775
          CLA       A   %BALANCE
          TYA       N     7770
GOTOSALT

CRR$
DO DONE FOR PAYMENT 1(1)N$
PRINT PAYMENT$
TAB$
INTEREST:BALANCE&MONRATE$
PRINCIPL:MONPAY-INTEREST$
PRINT PRINCIPL$
TAB$
PRINT INTEREST$
BALANCE:BALANCE-PRINCIPL$
TAB$
PRINT BALANCE$
DONE,CRR$
GO TO START$
END$
```

```
LOCATION  COMMAND   ADDRESS

          ORG       +3000
START     CRR
AREF      CLA       (BALANCE)
          TYA       +7770
BREF      CLA       (  $)
          TYA       +7764
          READY
          FST       BALANCE
          CLA       (RATE  )
          TYA       +7777
          READY
          FST       RATE
          CLA       (N  )
          TYA       +7774
          READY
          FST       N
          FCA       RATE
          FDV       (+12)
          FST       MONRATE
          FCS       N
          FST       STORE01
          FCA       MONRATE
          FAD       (+1)
          LN
          FMP       STORE01
          EXP
          FST       STORE01
          FCA       (+1)
          FSB       STORE01
          FST       STORE01
          FCA       BALANCE
          FMP       MONRATE
          FDV       STORE01
          FST       MONPAY
          CRR
          CLA       (MONTHLY )
          TYA       +7770
          CLA       (PAYMENT)
          TYA       +7777
          CLA       (+0006020+5705300
          STA       +2256
          FCA       MONPAY
          PRINT
          CRR
```

```
LOCATION  COMMAND   ADDRESS

          CRR
          CLA       (      )
          TYA       +7775
          CLA       (PAYMENT)
          TYA       +7770
          TAB
          CLA       (      PR)
          TYA       +7770
          CLA       (INCIPAL)
          TYA       +7777
          TAB
          CLA       (      INT)
          TYA       +7770
          CLA       (EREST)
          TYA       +7775
          TAB
          CLA       (      )
          TYA       +7775
          CLA       (BALANCE)
          TYA       +7770
          CRR
          FCA       (+1)
TAG 01    FST       PAYMENT
          FCA       PAYMENT
          PRINT
          TAB
          FCA       BALANCE
          FMP       MONRATE
          FST       INTEREST
          FCA       MONPAY
          FSB       INTEREST
          FST       PRINCIPL
          FCA       PRINCIPL
          PRINT
          TAB
          FCA       INTEREST
          PRINT
          FCA       BALANCE
          FSB       PRINCIPL
          FST       BALANCE
          TAB
          FCA       BALANCE
          PRINT
DONE      CRR
```

```
                FCA         PAYMENT
                FAD         (+1)
                FST         PAYMENT
                FSB         N
                TMI         TAG 01+00001
                TZE         TAG 01+00001
                TRA         START
                END
ALFCN01         BALANCE
ALFCN02           $
ALFCN03         RATE
ALFCN04         N
FLOCN01         +12
FLOCN02         +1
ALFCN05         .9,5,)6
ALFCN06         0-6.3,5
COMCN01         +0006020+5705300
ALFCN07
ALFCN08         PAYMENT
ALFCN09              PR
ALFCN10         8,:80-)
ALFCN11              INT
ALFCN12         343s5

START           +0000000-0030000
AREF            +0000000-0030010
BREF            +0000000-0030020
BALANCE         +0000000-0031100
RATE            +0000000-0031120
N               +0000000-0031140
MONRATE         +0000000-0031160
STORE01         +0000000-0031200
MONPAY          +0000000-0031220
TAG 01          +0000000-0030460
PAYMENT         +0000000-0031240
INTEREST        +0000000-0031260
PRINCIPL        +0000000-0031300
DONE            +0000000-0030620
ALFCN01         +0000000-0030670
ALFCN02         +0000000-0030700
ALFCN03         +0000000-0030710
ALFCN04         +0000000-0030720
FLOCN01         +0000000-0030730
FLOCN02         +0000000-0030750
ALFCN05         +0000000-0030770
ALFCN06         +0000000-0031000
COMCN01         +0000000-0031010
ALFCN07         +0000000-0031020
ALFCN08         +0000000-0031030
ALFCN09         +0000000-0031040
ALFCN10         +0000000-0031050
ALFCN11         +0000000-0031060
ALFCN12         +0000000-0031070
ENDTABLE        +0000000-0031320

END FIRST PASS
```

LOCATION	COMMAND	ADDRESS	
	ORG	+3000	L30000
START	TYC	+00370	
	TYC	+00100	+7200370+7200100
AREF	CLA	ALFCN01	
	TYA	+7770	+0030670+7277700
BREF	CLA	ALFCN02	
	TYA	+7764	+0030700+7277640
	TRA	+23100	
	TRA	C-00001	+5723100+5730030
	FST	BALANCE	
	CLA	ALFCN03	+3531100+0030710
	TYA	+7777	
	TRA	+23100	+7277770+5723100
	TRA	C-00001	
	FST	RATE	+5730051+3531120
	CLA	ALFCN04	
	TYA	+7774	+0030720+7277740
	TRA	+23100	
	TRA	C-00001	+5723100+5730100
	FST	N	
	FCA	RATE	+3531140+3031120
	FDV	FLOCN01	
	FST	MONRATE	+0530730+3531160
	FCS	N	
	FST	STORE01	+3431140+3531200
	FCA	MONRATE	
	FAD	FLOCN02	+3031160+0430750
	TRA	+27001	
	FMP	STORE01	+5727001+0731200
	TRA	+27201	
	FST	STORE01	+5727201+3531200
	FCA	FLOCN02	
	FSB	STORE01	+3030750+0631200
	FST	STORE01	
	FCA	BALANCE	+3531200+3031100
	FMP	MONRATE	
	FDV	STORE01	+0731160+0531200
	FST	MONPAY	
	TYC	+00370	+3531220+7200370
	TYC	+00100	
	CLA	ALFCN05	+7200100+0030770
	TYA	+7770	
	CLA	ALFCN06	+7277700+0031000
	TYA	+7777	

LOCATION	COMMAND	ADDRESS	
	CLA	COMCN01	+7277770+0031010
	STO	+2256	
	FCA	MONPAY	+6022560+3031220
	TRA	+22531	
	TYC	+00370	+5722531+7200370
	TYC	+00100	
	TYC	+00370	+7200100+7200370
	TYC	+00100	
	CLA	ALFCN07	+7200100+0031020
	TYA	+7775	
	CLA	ALFCN08	+7277750+0031030
	TYA	+7770	
	TYC	+00330	+7277700+7200330
	TYC	+00100	
	CLA	ALFCN09	+7200100+0031040
	TYA	+7770	
	CLA	ALFCN10	+7277700+0031050
	TYA	+7777	
	TYC	+00330	+7277770+7200330
	TYC	+00100	
	CLA	ALFCN11	+7200100+0031060
	TYA	+7770	
	CLA	ALFCN12	+7277700+0031070
	TYA	+7775	
	TYC	+00330	+7277750+7200330
	TYC	+00100	
	CLA	ALFCN07	+7200100+0031020
	TYA	+7775	
	CLA	ALFCN01	+7277750+0030670
	TYA	+7770	
	TYC	+00370	+7277700+7200370
	TYC	+00100	
	FCA	FLOCN02	+7200100+3030750
TAG 01	FST	PAYMENT	
	FCA	PAYMENT	+3531240+3031240
	TRA	+22531	
	TYC	+00330	+5722531+7200330
	TYC	+00100	
	FCA	BALANCE	+7200100+3031100
	FMP	MONRATE	
	FST	INTEREST	+0731160+3531260
	FCA	MONPAY	
	FSB	INTEREST	+3031220+0631260
	FST	PRINCIPL	

LOCATION	COMMAND	ADDRESS	
	FCA	PRINCIPL	+3531300+3031300
	TRA	+22531	
	TYC	+00330	+5722531+7200330
	TYC	+00100	
	FCA	INTEREST	+7200100+3031260
	TRA	+22531	
	FCA	BALANCE	+5722531+3031100
	FSB	PRINCIPL	
	FST	BALANCE	+0631300+3531100
	TYC	+00330	
	TYC	+00100	+7200330+7200100
	FCA	BALANCE	
	TRA	+22531	+3031100+5722531
DONE	TYC	+00370	
	TYC	+00100	+7200370+7200100
	FCA	PAYMENT	
	FAD	FLOCNO2	+3031240+0430750
	FST	PAYMENT	
	FSB	N	+3531240+0631140
	TMI	TAG 01+00001	
	TZE	TAG 01+00001	+5130461+5030461
	TRA	START	
	SL		+5730000+4000000
ALFCNO1	ALPHA	(BALANCE)	+7710710-1543401
ALFCNO2	ALPHA	$)	+5441041-0000000
ALFCNO3	ALPHA	(RATE)	+7520700-0441000
ALFCNO4	ALPHA	(N)	+7541020-0000000
FLOCNO1	DECIMAL	(+12)	+6000000-0000000
			+0000000-0000020
FLOCNO2	DECIMAL	(+1)	+4000000-0000000
			+0000000-0000001
ALFCNO5	ALPHA	(MONTHLY)	+6303100+2225220
ALFCNO6	ALPHA	(PAYMENT)	+3035360-0544000
COMCNO1	COMMAND	(+0006020+5705300)	+0006020+5705300
ALFCNO7	ALPHA	( )	-2041020-2000000
ALFCNO8	ALPHA	(PAYMENT)	+7660721+6013100
ALFCNO9	ALPHA	PR)	+7441020-2045450
ALFCN10	ALPHA	(INCIPAL)	-3143430+3034400
ALFCN11	ALPHA	INT)	+7441020-2063100
ALFCN12	ALPHA	(EREST)	-0520221+0000000
	END		

BALANCE $1200
RATE .09
N 24

MONTHLY PAYMENT 54.82

PAYMENT	PRINCIPAL	INTEREST	BALANCE
1.00	45.82	9.00	1154.18
2.00	46.17	8.66	1108.01
3.00	46.51	8.31	1061.50
4.00	46.86	7.96	1014.64
5.00	47.21	7.61	967.43
6.00	47.57	7.26	919.86
7.00	47.92	6.90	871.94
8.00	48.28	6.54	823.66
9.00	48.64	6.18	775.01
10.00	49.01	5.81	726.00
11.00	49.38	5.45	676.63
12.00	49.75	5.07	626.88
13.00	50.12	4.70	576.76
14.00	50.50	4.33	526.27
15.00	50.87	3.95	475.39
16.00	51.26	3.57	424.13
17.00	51.64	3.18	372.49
18.00	52.03	2.79	320.47
19.00	52.42	2.40	268.05
20.00	52.81	2.01	215.24
21.00	53.21	1.61	162.03
22.00	53.61	1.22	108.42
23.00	54.01	.81	54.41
24.00	54.41	.41	.00

6. CODING INFORMATION

6.1 FUNCTION TABLE: The SALT compiler contains a list of permissible function names in locations 2300 to 2327. Three function names -- SQRT, EXP, and LN -- must be included. The names for the other functions may be changed if necessary. The standard revised SALT compiler program recognizes the function names described in § 2.1.4, whose names are listed in 2300 - 2316.

Function names in this table are right justified; preceded by as many blanks as are necessary to make eight characters total. There must be a minus zero word after the list.

The SALT compiler does not itself generate any coding for the functions. The function name is punched out during the SALT phase, to be recognized as a SCRAP macro. The functions recognized by the standard revised SALT are defined as macros in the revised SCRAP. SCRAP must generate calling sequences to the subroutines which will evaluate the function.

6.2 ORIGIN: SALT sets up all program starting locations to be 3000.0. If for some reason it is desired to change this starting location, enter the new location in the following manner, before pressing Start 1.

(a) Set location counter to 0502.0
(b) Turn "fill tab" on typewriter down, and depress F key.
(c) Type + blank blank XXXX blank C/R, where XXXX is the new origin.

APPENDIX

TO

RECOMP II USERS' PROGRAM NO. 1034

PROGRAM TITLE:	SIGNAL CORP RECOMP ALGEBRAIC TRANSLATOR - SALT
PROGRAM CLASSIFICATION:	Executive and Control
AUTHOR:	H. D. Goddard Autonetics Industrial Products
	RECOMP II Users' Program No. 1034 has been modified to allow typing of SALT statements as rapidly as desired, initialize the SALT program for re-use and to permit processing of off-line tapes typed in SALT language.
CHECKED BY:	H. D. Goddard
PURPOSE:	The Signal Corps RECOMP Algebraic Translator (SALT) is a one pass compiler system which translates from algebraic statements into a SCRAP assembly program.
DATE:	July, 1961

Published by

RECOMP Users' Library

at

AUTONETICS INDUSTRIAL PRODUCTS
A DIVISION OF NORTH AMERICAN AVIATION, INC.
3400 E. 70th Street, Long Beach 5, Calif.

PROGRAM TITLE: SIGNAL CORP RECOMP ALGEBRAIC TRANSLATOR - SALT

RECOMP II Users' Program No. 1034 issued in July, 1961 permits the following:

I SALT statements may be typed as rapidly as desired. However typing speed is still restricted when typing SCRAP coding via an ENTER SCRAP statement.

II The SALT program may be initialized for re-use by depressing ERROR RESET and START 2. Thus it is not necessary to re-load the SALT tape if compiling a new program.

NOTE: After initializing for re-use, advance at least two folds of tape and depress START 1 if typewriter input is to be used or place paper tape in photoreader and depress START 3 if paper tape input is to be used.

III The SALT program can now process an offline prepared tape that has been typed in SALT language.

To use the SALT program and a tape prepared offline, the following is necessary:

1. Prepare the offline tape. The SALT input should be typed just as it would be on line.
   There should be at least eight blanks on the tape between each character. A Friden Flexowriter model FPC-5 modified according to RECOMP Technical Bulletin No. 21 will automatically insert blanks between each character.

2. Load and verify the SALT program. To verify, place the program tape in the photoreader past the zero memory section that is at the beginning of the tape.

3. Set tab override switch in the ON position.

4. Place offline prepared tape in photoreader.

5. Depress START 3.
   This will generate a SALT punched tape. Further processing (i.e. the generation of the 1st and 2nd passes in SCRAP) is the same as specified in RECOMP II Users' Program No. 1034 issued in January, 1961.

NOTE: If the SALT program detects an error while processing an offline prepared tape, it will do the following:

(a) If it is a SALT statement, it will print ERROR or PAIR ERROR followed by a print out of the statement.

(b) If it is an error in SCRAP coding (i.e. symbolic coding entered via an ENTER SCRAP statement), it will print SCRAP CODE ERROR. There will be no print out of the error.

NOTE: See RECOMP Technical Bulletin No. 21 for information concerning the preparation and use of a decimal data tape.