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

PURPOSE: The Signal Corps RECOMP Algebraic Translator (SALT) is a two 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. The SALT processor will accept fifteen (15) different statement formats. These allow for the necessary input/output operations, control functions, and processing functions.

DATE: January 1960

Published by

RECOMP Users' Library

at

AUTONETICS INDUSTRIAL PRODUCTS

A DIVISION OF NORTH AMERICAN AVIATION, INC.
3584 Wilshire Blvd., Los Angeles 5, Calif.

SIGNAL CORPS RECOMP ALGEBRAIC TRANSLATOR

by
T. J. TOBIAS, U..S. ARMY SIGNAL ENGINEERING AGENCY

INTRODUCTION: The Signal Corps RECOMP Algebraic Translator (SALT) is a two 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. The SALT processor will accept fifteen (15) different statement formats. These allow for the necessary input/output operations, control functions, and processing functions.

DESCRIPTION:

1. The SALT language contains four (4) types of statements as follows:

   a. ARITHMETIC STATEMENTS

   b. INPUT/OUTPUT STATEMENTS

   c. CONTROL STATEMENTS

   d. SPECIFICATION STATEMENTS

These statements are constructed from a set of key words in the format prescribed for the particular statement. In general, the statements have the following form:

tag, KEYWORD data $ or,

KEYWORD data $

For most types of statements, the use of a tag or statement number is optional. The specification statements and a few of the control statements are not tagged.

2. BASIC STATEMENT ELEMENTS: In addition to the key words, statements are constructed from the following basic elements:

   a. IDENTIFIERS: An identifier is an alphabetic word consisting of from one (1) to eight (8) alphabetic characters (A-Z). The single letter "C" should not be used. For some purposes the identifier is restricted to seven (7) characters in length. These abbreviated identifiers represent the name of a function, the name of an array, or the name of a subscript.

   b. VARIABLE: The name of a variable may be any identifier

which is not the name of a function. For example:

ALPHA

X

MATRIX

DATA

c. FUNCTIONS: A function has the following form, I(E), where I is an abbreviated identifier which has been reserved as the name of a function and E is an expression. For example:

SIN(E)

ARCTAN(E)

INTEGER(E)

SQRT(E)

d. NUMBERS: A number consists of the digits 0-9 and the decimal point "." . In SALT, a number may contain no more than fifteen (15) symbols, including the decimal point, and neither the integer nor the fractional part may contain more than eleven (11) digits. For example:

1

2.0

0.0039

.57

109.0336

e. SUBSCRIPTED VARIABLES: A subscripted variable may have one of the two following forms:

V(K) or,

V(I,J),

where V is the name of an array and I,J, and K are either numbers or variables. Examples of subscripted variables are as follows:

MATRIX(ROW, COLUMN)

TABLE(ITEM)

LIST(17,M)

VECTOR(21,3)

Z(9)

The following form is also permissible:

LIST(23.752)

MATRIX(7.013, 0.22)

However, since subscripts are inherently integer valued, the fractional parts will be disregarded so that the previous two examples would be interpreted as:

LIST(23) and,

MATRIX(7,0)

It shoud be noted that MATRIX(7,0) falls outside the defined area of the array which is named MATRIX.

f. EXPRESSIONS: An expression is the basic element of an arithmetic statement. An expression is defined as follows:

(1) A variable, a subscripted variable, a function, or a number is an expression.

(2) If X and Y are expressions and the expression Y does not begin with either "+" or "-", then the following are also expressions:

+Y	(+Y)	
-Y	(-Y)	
X+Y	X/Y	
X-Y	X&Y	(multiplication)
(Y)	X'Y	(exponentiation)
(X)		

Examples of expressions are as follows:

```
BETA&(X'3) + 23.019 - SIN(ALPHA)

(MATRIX(I,J)/ 13 ) &(ARCTAN( 1.770) ) - Y'(LOG(X/J))

A&X'2+B&X +C

(A+Z) / (N-1) - SQRT( 2&A)
```

3. STATEMENTS: The permissible SALT statements are as follows:

   a. INPUT/OUTPUT STATEMENTS:

      (1) READ The READ statement has the following format:

```
tag, READ variable $
```

      where the variable may be subscripted. For example:

```
01, READ X $

READ MATRIX (I,J) $

INPUT, READ TABLE( 12, 1) $

READ TABLE(ITEM, 7) $

23, READ DATA $
```

      (2) PRINT The PRINT statement has essentially the same form as the READ statement as follows:

```
tag, PRINT variable $
```

      where, as in the case of the READ statement, the variable may be subscripted. For example:

```
07, PRINT CUBEROOT $

PRINT MATRIX(P,Q) $
```

      (3) CXP The CXP statement is used as a check point during program test. It will cause the contents of the A and R registers to be printed as a floating point decimal number. The contents of A and R will not be disturbed. The statement has the following format:

```
tag, CXP$
```

For example:

```
45, CXP $

CXP $
```

b. ARITHMETIC STATEMENTS: In general, ARITHMETIC statements have the following form:

```
tag, variable: expression $
```

Examples of ARITHMETIC statements are as follows:

```
05, X: X-1 $

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

COMP, ELEMENT(5) : ELEMENT(5)/ELEMENT(J) + 21.776 $

GROUP : GROUP + 1 $
```

An ARITHMETIC statement may contain any expression that is permissible under the rules outlined in the definition of an expression.

c. CONTROL STATEMENTS:

(1) GO TO: The GO TO statement causes an unconditional change of control to the named statement. In general, the GO TO statement has the following form:

```
tag, GO TO label $  or

tag, GOTO  label $
```

For example:

```
12, GO TO 09 $

GOTO ANYPLACE $

ENDOFAA, GO TO 17 $
```

(2) IF: The IF statements will cause a conditional change of control depending on the value of an expression or the setting of a sense switch. The IF statements have the following form:

```
tag,  IF(expression) minus, zero, plus  $

tag,  IF(SENSE n ) on, off $
```

For example:

```
23, IF(X- 0.12) 01, INPUT, 17  S

IF(X'2 + SIN(ALPHA) - GAMMA) 21, 02, OUT $

ANYOUT,  IF(SENSE C) 23, ENDOFB  $
```

(3) DO: The DO statement is used to control iteration loops. The controlled variable may be a subscript or a variable. The DO statement has the following form:

```
tag,  DO label FOR index start (delta) limit $
```

For example:

```
55, DO 56 FOR X 1(1)  10  $

DO MOVE FOR K 1(1) M $

DO 01 FOR Y P(Q)R  $
```

Examples of the use of DO statements is contained in paragraph 4.

(4) STOP: The STOP statement represents a dynamic end of a program. This statement has the following form:

```
tag, STOP  $
```

For example:

```
99, STOP$

ENDOFIT ,  STOP  $

STOP  $
```

(5) CONTINUE: The CONTINUE statement is a dummy statement which generates no object code. The primary use of this statement is as the last statement of a DO range. This statement has the following form:

```
tag, CONTINUE  $
```

(6) RETURN: The RETURN statement marks the dynamic end of a subroutine. It is usually paired with a ROUTINE statement. The RETURN statement has the following form:

tag, RETURN name $,

where the field "name" is the tag used at the beginning of the subroutine. For example:

```
ROUTINE CUBEROOT $

  .
  . (subroutine)
  .

81, RETURN CUBEROOT $

ROUTINE XYZ $

  .
  . (subroutine)
  .

RETURN XYZ  $
```

(7) END: The END statement marks the end of the data to be assembled. This statement is not tagged. It has only the following form:

```
END $
```

(8) PAUSE: The PAUSE statement will cause a SCRAP PAUSE to be punched on paper tape. This statement is not tagged. It has only the following form:

```
PAUSE  $
```

(9) ENTER SCRAP: The ENTER SCRAP statement will cause the SALT processor to allow the input of SCRAP coding. This statement has only the following form:

```
ENTER SCRAP $
```

The processing of SALT statements may be continued by use of the location "GOTOSALT" in the SCRAP coding. The SALT processor will then process SALT statements.

d. SPECIFICATION STATEMENTS:

(1) ROUTINE: The routine statement marks the dynamic beginning of a subroutine. This statement is not tagged. It has the following form:

```
ROUTINE name  $
```

where, "name" is the name of the subroutine. This

statement will cause the construction of only the simplest type of subroutine calling sequence and only one exit. Examples of the use of the ROUTINE statement have been given under the discussion of the RETURN statement.

(2) ARRAY: The ARRAY statement must be given for each subscripted variable. This statement will reserve the necessary storage area for the array and will construct the appropriated constants needed for the processing of subscripted variables. In general, the ARRAY statements have the following form:

ARRAY name (items) S or

ARRAY name (rows, columns) $

The dimensions of the array must be numeric data. For example:

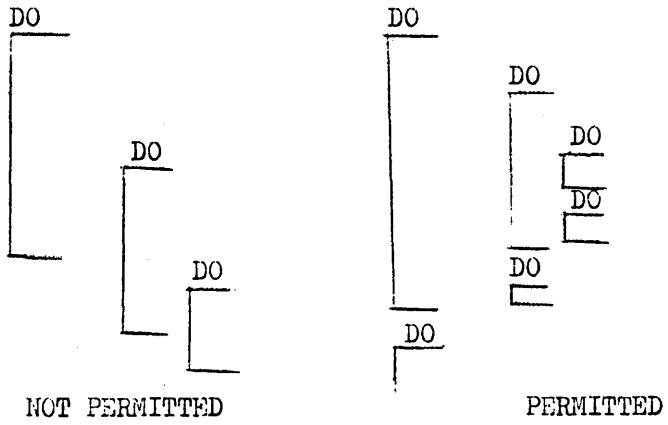
ARRAY MATRIX (10,10) S

ARRAY TABLE (5,8) S

ARRAY LIST (25) $

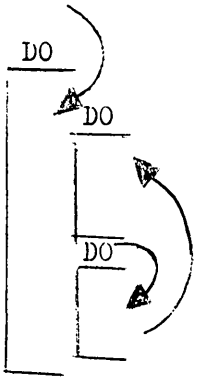
4. USE OF DO STATEMENTS: The DO statements require that certain rules be followed so that proper indexing will occur. The three rules which must be observed are as follows:

a. If a DO statement is contained in the range of another DO statement, all statements in the range of the second DO must be contained in the range of the first DO statement. For example:

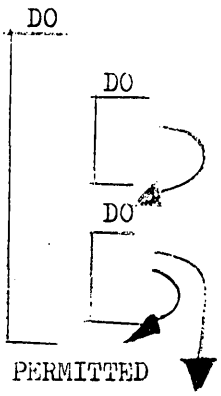


NOT PERMITTED PERMITTED

b. No transfer of control is permitted into the range of DO from outside the range of the DO statement.



NOT PERMITTED



PERMITTED

c. The last statement in the range of a DO may not be an IF or a GOTO statement. In such cases the CONTINUE statement is used as dummy. For example, the following is a Table Look-up Operation for a table containing 100 items:

```
DO 20 FOR K 1(1) 100
20, IF  (LIST(K) - ARG ) 20,21,20 $
21,
```

NOT PERMITTED

```
DO 20 FOR K 1 (1) 100 $
IF (LIST(K) - ARG ) 20,21,20 S
20, CONTINUE $
21,
```

PERMITTED

5. OPERATION: The SALT statements are entered into the SALT processor from the RECOMP typewriter in the format prescribed in paragraph 3. The SALT processor translates each statement as it occurs and punches the interpretation of the statement in SCRAP language into paper tape. The translated program may then be assembled by the SCRAP assembly program. The program to be compiled may, in addition, contain SCRAP language coding. These lines of coding are not processed but are copied directly into paper tape. The SALT processor performs a check for correct statement construction. In those statements which contain expressions, the expression is examined to determine if it is properly constructed. Such SCRAP coding as may be entered is also edited for proper construction.

SUMMARY: The SALT processor is a two pass compiler system which will translate algebraic statements into SCRAP assembly language.

The SCRAP language program may then be assembled into a final object program. The SALT language contains fifteen (15) statements which will allow for the expression of many numerical processes and which will provide a convenient and rapid means of translation from a problem language into machine coding.

```
01, RESULT : (DATA'2 - 1)&(SQRT(ALPHA - GAMMA/DELTA)) + IOTA $

01, STOP $

    END $
```

LOCATION	COMMAND	ADDRESS
	ORG	+0500
LINE 01	FCA	GAMMA
	FDV	DELTA
	FST	STORE01
	FCA	ALPHA
	FSB	STORE01
	FST	STORE01
	FSQ	STORE01
	FST	STORE01
	FCA	DATA
	FMP	DATA
	FSB	(+1)
	FMP	STORE01
	FAD	IOTA
	FST	RESULT
LINE 02	HALT	C
	END	
FLOCNO1	+1	

LOCATION	COMMAND	ADDRESS
	ORG	+0500
LINE 01	FCA	GAMMA
	FDV	DELTA
	FST	STORE01
	FCA	ALPHA
	FSB	STORE01
	FST	STORE01
	FSQ	STORE01
	FST	STORE01
	FCA	DATA
	FMP	DATA
	FSB	FLOCNO1
	FMP	STORE01
	FAD	IOTA
	FST	RESULT
LINE 02	HALT	C
	SL	
FLOCNO1	DECIMAL	(+1)
	END	

LINE 01	+0000000-0005000
GAMMA	-0000000-0000000
DELTA	-0000000-0000000
STORE 01	-0000000-0000000
ALPHA	-0000000-0000000
DATA	-0000000-0000000
IOTA	-0000000-0000000
RESULT	-0000000-0000000
LINE 02	+0000000-0005070
FLOCNO1	+0000000-0005100
ENDTABLE	+0000000-0005120

LINE 01	+0000000-0005000
GAMMA	+0000000-0005120
DELTA	+0000000-0005140
STORE01	+0000000-0005160
ALPHA	+0000000-0005200
DATA	+0000000-0005220
IOTA	+0000000-0005240
RESULT	+0000000-0005260
LINE 02	+0000000-0005070
FLOCNO1	+0000000-0005100
ENDTABLE	+0000000-0005300

```
END FIRST PASS
```

```
L05000
+3005120+0505140
+3505160+3005200
+0605160+3505160
+4405160+3505160
+3005220+0705220
+0605100+0705160
+0405240+3505260
+7705070+4000000
+4000000-0000000
+0000000-0000001
```

Sample of SALT Coding

RECOMP II USERS' PROGRAM NO. 1034

APPENDIX 1

PROGRAM TITLE: SIGNAL CORPS RECOMP ALGEBRAIC TRANSLATOR - SALT OPERATING INSTRUCTIONS

PROGRAM CLASSIFICATION: Executive and Control

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

PURPOSE: The SALT processor accepts statements which are typed from the RECOMP typewriter and translates these into SCRAP symbolic instructions which are then punched into paper tape. This document contains the information concerning the following:

a. Operating Procedure

b. Permissible Functions

c. Restrictions and Error Halts

DATE: February, 1960

SIGNAL CORPS RECOMP ALGEBRAIC TRANSLATOR, SALT

OPERATING INSTRUCTIONS

1. GENERAL: The SALT processor accepts statements which are typed from the RECOMP typewriter and translates these into SCRAP symbolic instructions which are then punched into paper tape. This document contains the information concerning the following:

   a. Operating Procedure

   b. Permissible Functions

   c. Restrictions and Error Halts

2. OPERATING PROCEDURE: The SALT program accepts input from the RECOMP typewriter and produces SCRAP instructions of paper tape. The SALT processor checks for correct statement construction and when SCRAP coding is entered the identical checks are made as are performed by the SCRAP processor. The operating procedure for SALT is as follows:

   a. Initial Operations:

      (1) Load SALT program tape.

      (2) Set typewriter margin at 10; tabs at 20, 29, and 32.

      (3) Depress START 1 to begin.

   b. Typing STATEMENTS:

      (1) Type each statement being careful to insure that each number or name is terminated by a space, figures shift, letters shift, or carriage return. These four functions serve as end of field marks. The symbols +, -, &, /, (,), ', :, $, ', are individually recognized when they are the first symbol of a new field but not when contained within a field. For example, $\ldots\&{}^{L}_{S}\mathrm{DATA}^{F}_{S}))+{}^{L}_{S}\mathrm{ALPHA}^{F}_{S}+\ldots$ is correct. However, $\ldots{}^{F}_{S}137.2)^{L}_{S}$ or ${}^{F}_{S}01,{}^{S}_{P}$ are not correct, since in each case the numeric field has not been terminated properly. The following would be correct: ${}^{F}_{S}137.2^{F}_{S})$ or ${}^{F}_{S}137.2^{LF}_{SS})$ or 01^{S}_{P}, or 01^{F}_{S}, .

      Typing extra spaces, letters or figures shifts, or carriage returns is permitted. The Tab function is also permitted.

(2) Each statement is terminated by use of the $ code. It serves as an "enter" function. If format errors are detected in statement before the $, the word "ERROR" will be typed. The line must be re-entered correctly.

(3) If an error is discovered while typing, the statement may be deleted by striking the blank, 02, key next to the "M" key. The statement must be re-typed.

(4) If an output error is caused by typing too quickly, this condition may be corrected by depressing

(a) Error Reset, then

(b) START 1, and then

(c) Re-type the line

(5) Entering of SCRAP coding is accomplished in the same manner as specified in the SCRAP Operating Instructions.

c. Termination: The end of the program must be signified by use of the statement, END $.

3. PERMISSIBLE FUNCTIONS: The SALT processor contains a list of permissible functions in locations 2300 - 2327. This provides space for 23 functions. Three of the function names, SQRT, EXP, LN must be included. The names for the other functions may be changed if desired. The standard SALT program contains the following list of functions:

SQRT	LOGTEN
INTEGER	LOGE
SIN	LOG
COS	LN
TAN	ABS
ARCTAN	EXP
ARCSIN	TENPOW
ARCCOS	TWOPOW
LOGTWO	

The list must be terminated by a minus zero word and the function names may not exceed seven characters in length.

The calling sequence for the subroutine is indicated by the name of the function as the op code in the SCRAP coding. For example,

The SALT input; 07, DATA : SIN (ALPHA) $

would generate

```
LINE 07   FCA    ALPHA
          SIN
          FST    DATA
```

as SCRAP coding. The exact calling sequence must be generated by the SCRAP assembly program with the name of the function, "SIN", serving as the name of the macro instruction. If the sine routine is in location 0100.0 and if the argument is provided to it in A,R and the sine of the argument is produced in A,R, then the SCRAP assembly program must substitute a TRA 0100 for the SIN line of coding. Thus producing as a result of the first pass:

```
LINE 07   FCA    ALPHA
          TRA    0100
          FST    DATA
```

The SALT processor assumes that all of the function subroutines will have the following standard calling sequence and specification:

a. Functions must be of only one argument and produce only one result.

b. Argument and result will be in floating point.

c. Function subroutine, I, will produce I(W) in the A and R, where the argument, W, was in A and R at subroutine entry.

d. Calling sequence:

```
          FCA L(W)*
          TRA L(I)          C(A,R) = W
+1        RETURN            C(A,R) = I(W)
```

*May be any series of operations which leaves W in A and R. This coding is provided by SALT except when instructions are inputed in SCRAP language.

e. When subroutine calls are automatically compiled, it is most convenient if "error" conditions are resolved, if possible, within the routine without error returns. However error returns may be compiled if provided in the SCRAP macro-instruction definition of the calling sequence.

4. RESTRITIONS AND ERROR HALTS:

a. An error in the construction of a statement will be indicated by the printing of the word, ERROR.

b. Incorrect construction of an arithmetic statement will produce the two words, PAIR ERROR.

c. When an error is detected, the statement is erased and the SALT processor returns to the beginning of statement mode.

d. The only restriction on statement construction, other than those imposed by format, is that no statement may exceed 128 elements, an element being a name, a number, or a symbol. This condition may only occur for a statement containing an arithmetic expression. If the maximum is exceeded the word ERROR will be printed.

C4