503 TECHNICAL MANUAL

VOLUME 2 : PROGRAMMING INFORMATION

PART 1 : PROGRAMMING SYSTEMS

SECTION 2 : SYMBOLIC ASSEMBLY

CHAPTER 5

(Chapters 1-4 inclusive have already been issued on a provisional basis)

Elliott Computing Division, Elliott Bros. (London) Ltd.
Elstree Way, Borehamwood, Herts.

May, 1963.

CHAPTER 5.

"REPLACE" AND "MODIFY" STATEMENTS

1) Replacement Statements: a general description

The sections before this point describe the use of identifiers to find Symbolic Assembly Code programs in the store of the 503, and for this purpose an identifier is used to single out a particular, but unknown, location in the store. This chapter describes a way in which whole sections of symbolic assembly code may be identified, so that they can afterwards be assembled together into a program.

A section of symbolic assembly code of this type will be called a "replacement-text", and the identifier associated with it a "replacement-identifier". The Symbolic Assembly Program will always replace the identifier with the appropriate text, if the identifier is given as part of the input program to be assembled. (The previous chapter gave a description of the replacement-text 00 LINK/40 1, for which the replacement-identifier was EXIT.)

This facility can also be used to modify and correct existing Symbolic Assembly Code tapes, because a replacement-identifier can also be an identifier already in use in the same program, if desired. This makes it possible to modify a program at any point where its tape makes use of an identifier. However, an identifier will not, in general, pin-point a unique point on the tape, as it will almost certainly be given again somewhere else in the program.

It is not essential to specify a replacement-text when its replacement-identifier is introduced: in certain circumstances it is sufficient to give the source of the text, so that it can be found when the identifier is called by name. This source could be a second reader, for example. Hence it is possible to use tapes of standard functions, each function being specified in the form of a replacement-text.

Finally, there is a parameter facility which makes it possible to specify parts of a replacement-text at the time when the text is assembled into the program.

2) Form in which replacements are specified

Replacements are specified in two stages. These are:-

(1) Introduction of the identifier by which the replacement is to be known, and

(2) Specification of the text of the replacement.

The text does not have to be specified at Stage 1; however, it must be at hand when required for insertion into a program.

A replacement is called whenever SAP reads its identifier while assembling a program. This makes SAP search for the appropriate replacement-text. When this is found, it is input, and SAP then passes on to the next code instructions. The identifiers used within a replacement-text assume, in general, the meanings which are current when the replacements are made.

3) Replacement Introductions, their form

A replacement is introduced by giving its identifier and the source of its text, (so that the text may be found when needed). The sources recognised at the moment are,

R	-	the second reader
S	-	the store
T	-	the typewriter
Z	-	no source

Introductions are effected by listing the replacement identifiers in the usual way, after the basic name replace, each identifier being followed by an equals-sign and the letter which corresponds to the source of its text.

e.g. replace PRINT=R, INPUT=R, Mod 17=T;

If the source is given as S (the Store), the replacement text must follow immediately, enclosed in diamond brackets. The source, if unspecified, is understood to be S.

e.g. replace exhort ⟨...text of exhort...⟩ ;

A replacement-identifier may be associated with a different text by re-introducing it, or its use as a replacement-identifier may be stopped altogether, by introducing it as belonging to source Z.

Certain identifiers are so positioned that they can only be interpreted as replacement-identifiers: for example, identifiers on lines by themselves. These are ignored by SAP if they are unintroduced when read. In this respect also, replacements introduced as belonging to source Z are considered to be unintroduced.

Replacements should not be introduced outside SAC programs.

4) Replacement-Texts. Parameters

Any section of SAC may be used as a replacement text; this includes anything from a single identifier or a function to a complete section of SAC, including data-, label-, (and even replacement-) introductions etc. (But this text should not begin or end with any incomplete introduction or instruction). As

Replacement-Texts.
Parameters (continued)

present in the source from which it is to be drawn, a replacement-text should invariably be enclosed within diamond brackets, and preceded by the appropriate replacement-identifier.

The text employs identifiers of 3 different types. These will be referred to in later sections as Type A, Type B, and Type C. They are:

Type (A) Label, data and block identifiers. These have been described in previous chapters.

Type (B) Replacement-identifiers. Replacements may call each other to any depth, (though a replacement cannot call upon itself, since this would lead to an infinite recursion - see Section 6).

Type (C) Parameter-identifiers; these are identifiers which may be "re-named" at the time of inserting the replacement into a program. A parameter-identifier could be renamed as -

(i) an integer

or (ii) a constant enclosed within diamond brackets; which is equivalent to the address of a location containing the constant thus enclosed

or (iii) another identifier, not itself a parameter.

"Parameter-identifiers" will be subsequently referred to as "formal-parameters", and their permitted renamings or replacements will be termed "actual-parameters".

The formal parameters of a replacement, if any, are listed with and just before its text. A semicolon should be used to separate them from the text.

For example, a text might be given as,

```
COPY ⟨reader, writer;
      again) 060  : 71 reader
             20 work / 74 writer
             40 again⟩
```

in which "reader" and "writer" are the formal parameters.

The separating semicolon must always appear just before the replacement text, <u>even if there are no parameters</u>.

The actual parameters are given at the time of calling the replacement into the assembled program. This is done by listing them just after the replacement— identifier, enclosed in round brackets, and in the order set up by the formal-parameter list.

Replacement-Texts.
Parameters (continued)

Hence COPY (2048, 4096)

is equivalent to

```
again) 060 : 71 2048
             20 work / 74 4096
             40 again
```

using the previous example.

5) Interpretation of "Ambiguous" Identifiers

(i) The "order of precedence" set up among the identifiers is
Type C before Type B before Type A

The "meaning" or reference, of an identifier which belongs to more than one of the groups A B and C may be found by referring to this order of precedence. For example, a formal parameter will always be interpreted as such, even though its identifier is also used as a replacement-identifier or a label-identifier.

Thus, replacement-statements do not affect formal-parameters; but they do affect actual-parameters which are also identifiers.

For example, (as might be expected) the effect of the introduction replace whichone ⟨thatone⟩ , thatone⟨⟨+1⟩⟩ , thisone⟨⟨+2⟩⟩

GET ⟨whichone; 30 whichone⟩ ;

on the instruction

GET (thisone)

is to give 30 ⟨+2⟩

The identifier "whichone" has been used as both a replacement-identifier and a parameter: its use as a formal-parameter takes precedence over its use as a replacement-identifier.

Thus, on calling the replacement GET, its formal-parameter "whichone" becomes the actual-parameter "thisone"; and "thisone" identifies the replacement "thisone" +2 , so that the total effect of

GET ⟨thisone⟩

is 30 ⟨+2⟩

(ii) A formal parameter only operates within its own replacement text; it will not operate within any replacement which has been rested inside that replacement.

Example, the effect of

replace A ⟨B; 10 B:20 C⟩ , B ⟨ ; E⟩ , C ⟨ ; B⟩ ;

on the instructions

```
30 B
A(D)
```

is to give

```
30 E
10 D:20 E
```

A's formal parameter, B, has no effect outside A's text, so that the identifier in

```
30 B
```

is only operated on by the replacement

B ⟨;E⟩ , to give

```
30 E.
```

A(D) is replaced by 10 B: 20C . B, being a formal parameter, becomes the actual parameter D. C is affected by the replacement statement C ⟨; B⟩ , but the text of this replacement, B, is not a reference to the formal parameter B, but to the replacement B. The total effect on this order-pair is thus

```
10 D : 20 E
```

(iii) A replacement's identifier could be quoted in its own text, either explicitly, or else as part of the text of one of the rested replacements. This would be taken as a reference to some parameter - , data-, label-, or block-element.

6) The Different Sources

(i) Section 3 outlined the way in which replacements could be introduced to different sources. If this source was given as the typewriter, then

insert "identifier"

will be output on the typewriter whenever "identifier" is called as a replacement, and SAP will wait for the text to be typed in. This text may vary from occasion to occasion. SAP will take "identifier" itself as the text if the operator types the character =

Incorrectly typed texts may be cancelled by typing the non-escapable character , but the text has then to be input from its beginning.

The Different Sources(continued)

(ii) If the identifiers A, B, ... have been introduced as replacements, which are to be input from the same source, then the instruction,

A, B, ...

on a new line causes them to be searched for, and input in the order in which they are found.

This means that a functions-tape may contain the replacements A,B,C,... in any order, so that the tape need only be read once, and all the replacements will be searched and input.

7) Modify Statements

The effect which a Replace-Statement has on a SAC program is that the Replacement-text is substituted for the Replacement-identifier, wherever the latter occurs in the program.

Modify Statements are similar, except that:

(i) Whole sections of program may be deleted when the modification is made

and (ii) Modifications are only applied to labelled sections of program.

Thus 3 bits of information should be specified in a modification introduction.

(a) The modification-identifier. This is always the name of a label marking the position in the program at which the text is to be inserted.

(b) The number of instructions to be deleted from the program by the modification. This number may be zero; if greater, it is taken to include the instruction labelled by the modification-identifier.

When counting the number of instructions to be deleted by a modification, take whole words other than order-pairs as single instructions. Labels belong to the instructions which they identify, and the control names begin, end, data, together with the lists of identifiers which follow them, are also counted as single instructions.

(c) The source from which the modification-text is to be drawn.

Replace and Modify-Statements are identical in all other respects.

Modify Statements (continued)

Example: the introduction

```
 modify    L1)0 ⟨;04 A⟩ ;
```

changes the text

```
            30 B
          L1)20 C
```

to

```
            30 B
          L1)04 A
            20 C
```

Note that the modification identifier is followed by a closed bracket in these introductions.

Example : the introduction

```
modify    L1)1 ⟨ ;L2) 04A : 20 D⟩ ;
```

changes the text

```
                        30B : L1)20 C
```

to

```
                                 30 B
                         L2)04A : 20 D
```