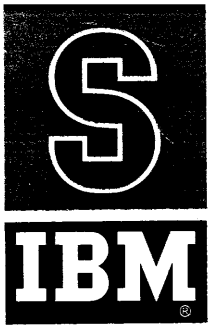

7030 DATA PROCESSING SYSTEM BULLETIN

SMAC Language Processor

INTRODUCTION

The Stretch MACro language, SMAC, can be freely mixed with STRAP. This write-up describes the SMAC language, the processor for SMAC, and the SMAC Generator Language.

A knowledge of STRAP is presumed.

SMAC LANGUAGE

A SMAC program is a mixture of SMAC and STRAP statements. A SMAC statement consists of the following parameters: a card code, a tag, an operation, and arguments. The arguments of a SMAC statement are literals or symbolic addresses plus or minus an absolute increment and modified, if necessary, by an absolute index register. Data capabilities in a SMAC program are those of STRAP.

SMAC Statement Card Format

Only one SMAC statement may appear on a card. The card format is:

a. Column 1: may contain a comment mark or a continuation mark as in STRAP.

b. Columns 2-9: tag field.

c. Columns 10-72: variable field. These columns have the form

$$OP, ARG_1, ARG_2, \ldots, ARG_N$$

OP consists of up to nine alphabetic characters the first of which is M and none of which are Q. ARG_i can be a literal or a symbolic address plus or minus an absolute increment and modified by an absolute index register. Tags and symbolic addresses can be a maximum of eight alphanumeric characters, the first of which is alphabetic and none of which is Q. Operations and symbolic addresses cannot be split across card boundaries. Tags and symbolic addresses in STRAP statements may not contain Q.

The variable field of a continuation card is in columns 10-72. No other columns are examined by the processor.

If the preceding card contains a comment mark in columns 10-72, the continuation card is a continuation of the comment field. A maximum of nine continuation cards is possible. A card with a comment mark in column 1 cannot be continued.

d. Columns 73-80: not used by the processor.

SMAC Operations

Except for a very basic set, the operations in SMAC are specified by the user. A library tape and a library updating program are available so that new operations and their generators can be added at any time. The library updating program adds the generator to the library tape and updates the table of acceptable SMAC operations.

PROCESSOR

The processor for the SMAC language, which requires only one pass over the source program, consists of a processor shell, a branch table, processor subroutines, and generators.

Processor Shell

The processor shell determines the sequence in which subroutines and generators are executed to process a given SMAC program. SMAC statements are processed one at a time. Reading and identification of each statement is done by subroutines, after which the generator for a particular statement takes control. When the generator has been executed, the processor shell resumes control. There is an N to 1 relationship between SMAC statements and generators where $N \geq 1$. That is, more than one SMAC statement may result in the calling of the same generator.

Branch Table

The branch table consists principally of branches to subroutines, control information, and locations of buffers.

The relative positions of entries in the branch table remain fixed. Thus the relocation of a subroutine or a buffer requires only the changing of the absolute address of the subroutine or buffer in the branch table, and not the reassembly of any other part of the program. In order that the branch table itself may be relocated, all references to branch table entries are modified by Index Register 2, which contains the branch table address.

Processor Subroutines

Processor subroutines include I/O routines, scans, and list routines. Entry to all subroutines or access to buffers is made through the branch table.

Generators

Generators are relocatable routines and may be roughly classified as being of two types: a generator which produces a sequence of object code for a given SMAC statement and a generator which performs some prescribed function when a given SMAC statement is encountered, but which does not produce any object code.

A generator of the first type consists of two parts: logic and skeletons. The logic is written in SMAC Generator Language; the skeletons are the coding which the generator substitutes for a given SMAC statement. Skeletons may contain both SMAC and STRAP statements. The skeleton portion of a generator follows the logic portion and consists of one or more skeletons. The selection of one or more of the skeletons is made by the logic portion.

SMAC GENERATOR LANGUAGE

Certain SMAC operations are specified as SMAC Generator Language statements. These statements in no way differ from regular SMAC statements as described in the SMAC Language section. However, the generators for the Generator Language statements are stored on a special library tape which is called in only when generators are being compiled. Statements can be added to the Generator Language in the same way that statements are added to the SMAC language.

The Generator Language presently consists of the statements outlined below. A SMAC statement consists of a series of parameters where:

column 1 is a parameter,
columns 2-9 are a parameter,
the operation field is a parameter, and
each argument is a parameter.

Five pointers, whose symbolic tags are QPT1, QPT2, QPT3, QPT4, and QPT5, may be used to point at any parameter of the subject statement. Parameters are numbered relative to a given pointer. The parameter pointed at is parameter 1 and succeeding parameters are numbered 2 through n.

Examples:

1.	subject statement:		QPT1 col. 1	TAG	MOP,	arg_1,	arg_2,	arg_3,	arg_4
	parameter numbers:		1	2	3	4	5	6	7
2.	subject statement:	col. 1	TAG	MOP,	QPTi arg_1,	arg_2,	arg_3		
	parameter numbers:				1	2	3		

Generator Language Statements

List of Generator Language Operations

Operation
MBGEN
MEXIT
MBSKL
MEGEN
MMVPT
MOUT
MGETL
MKPEQ
MKPNEQ
MKCEQ
MKCNEQ
MOUTF
MOUTY

Functions of Generator Language Statements

1. MBGEN
   (Begin GENerator)

This statement must be the first statement of a generator.

It places the count of the number of parameters of the subject SMAC statement in the count field of an index word in the branch table whose symbolic tag is QCOUNT.

It initializes the pointer, whose symbolic tag is QPT1, to point at the first parameter (i.e., column1) of the subject SMAC statement.

2. MEXIT

This statement must be the last statement of a generator to be executed. It returns control to the processor shell.

3. MBSKL
   (Begin SKeLeton)

This statement is used in the skeleton portion of a generator to indicate that the statements which follow are to be put in skeleton format, i.e., STRAP Data Definition statement with IQS entry mode.

A skeleton is terminated by another MBSKL statement or the MEGEN statement.

4. MEGEN
(End GENerator)

This statement must be physically the last statement of a generator.

This statement inserts all the SYN cards necessary for branch table reference.

5. MMVPT, QPTi, N, QPTj
(MoVe PoinTer)

This statement allows the generator programmer to create additional pointers to point at particular parameters of the subject SMAC statement.

QPT1 is initialized by the MBGEN statement which sets it to point at column 1. Four additional pointers whose symbolic tags are QPT2, QPT3, QPT4, and QPT5 may be created by using the MVPT statement, where QPTi is any previously defined pointer, N is the positive number of parameters by which the programmer desires to move the pointer along the subject statement, and QPTj is the new pointer.

QPTi is not destroyed unless QPTi equals QPTj, e.g., MMVPT, QPT1, 3, QPT1.

Example:

subject statement before pointer is moved: QPT1 col. 1 TAG OP, arg_1, arg_2, arg_3, arg_4

move pointer statement: MVPT, QPT1, 3, QPT2

subject statement after pointer has been moved: QPT1 col. 1 TAG QPT2 OP, arg_1, arg_2, arg_3, arg_4

N may also be QCN, where QCN is a value field in the branch table in which the generator programmer may store a computed value for N prior to encountering the MMVPT statement.

6. MOUT, QPTi, $NAME_1$, $NAME_2$, ..., $Name_n$
(OUTput)

This statement may be used to output the skeletons indicated by $NAME_i$ where parameters are inserted in the skeletons according to the parameter numbers determined by the pointer.

Example:

SMAC statement

```
NAME    MADD,A,B,C
```

Generator

```
            MBGEN
            MOUT, QPT1, SK1
            MEXIT
SK1         MBSKL
/2/         L, /4/
            +, /5/
            ST, /6/
            MEGEN
```

output

```
NAME        L, A
            +, B
            ST, C
```

7. MGETL, QPTi, N
   (GET Literal)

This statement allows the programmer to access parameters of a subject statement known to be literals. The literal indicated by the pointer and the parameter number is converted to a binary integer and placed in the value field in the branch table whose symbolic tag is QLIT.

8. MKPEQ, QPTi, N, AB..., TAG1
   (Compare Parameter EQual)

   MKPNEQ, QPTi, N, AB..., TAG1
   (Compare Parameter Not EQual)

Either of these statements allows the generator programmer to compare a whole parameter against any desired set of eight or fewer characters, and to transfer control to TAG1 depending on the result of the comparison.

QPTi is a pointer, N is a parameter number, AB... is the set of one to eight characters (commas and blanks may not be used) to be compared with the parameter indicated, and TAG1 is the statement to which control may be passed. If the parameter contains more characters than specified, the result is not equal.

MKPEQ transfers control to TAG1 if the result of the comparison is an equality.

MKPNEQ transfers control to TAG1 if the result of the comparison is an inequality.

Example:

Subject statement: OP, R23 ?T, FGH, IJKL

MKCEQ, QPT1, 4, R23?T, NAME
Result: equal
MKPEQ, QPT1, 5, FG, NAME
Result: unequal

9. MKCEQ, QPTi, N, AB..., TAG1
(Compare Characters EQual)

MKCNEQ, QPTi, N, AB..., TAG1
(Compare Characters Not EQual)

Either of these statements allows the programmer to compare the first one to eight characters of a parameter with an equal number of characters specified in the compare character statement.

QPTi is a pointer, N is a parameter number, AB... is the set of one to eight characters to be compared with an equal number of characters of the parameter indicated, and TAG1 is the statement to which control may be passed.

MKCEQ transfers control to TAG1 if the result of the comparison is an equality.

MKCNEQ transfers control to TAG1 if the result of the comparison is an inequality.

Example:

Subject statement: OP, R23?T, FGH, IJKL

MKCEQ, QPT1, 4, R23?T, NAME
Result: equal

MKCEQ, QPT1, 5, FG, NAME
Result: equal

10. MOUTF, SK1, n_1, SK2, n_2, . . . , SK_m, n_m
(OUTput Fixed number of parameters)

This statement is used in a generator to select a certain skeleton from a group, depending upon the number of arguments in the SMAC statement. SKi is the name of a skeleton and n_i is the number of arguments which will cause skeleton SKi to be put out. There may be as many as 15 different selections. Parameters of the subject statement are numbered according to a pointer which points at column one.

This statement may be used in a generator for one particular SMAC operation where different skeletons are put out depending upon the number of parameters. It may also be used in a general type of generator which is called by several different SMAC operations whose skeletons can be distinguished on the basis of the number of arguments in the SMAC statement. This latter use is demonstrated in the following example:

SMAC statement 1 which always contains three arguments:

```
NAME    MOP1, A1, A2, A3
```

Skeleton:

```
SK1     MBSKL
/2/     L, /4/
        *, /5/
        ST, /6/
        B, $/3/
```

SMAC statement 2 which never has any arguments:

```
NAME    MOP2
```

Skeleton:

```
SK2     MBSKL
/2/     B, $SUBRT
          , /3/
```

SMAC statement 3 which always contains five arguments:

```
NAME    MOP3, C1, C2, C3, C4, C5
```

Skeleton:

```
SK3     MBSKL
/2/     L, /4/
        +, /5/
        -, /6/
        *, /7/
        ST, /8/
```

SMAC statement 4 which always contains three arguments:

```
NAME    MOP4, D1, D2, D3
```

Skeleton:

```
SK1     MBSKL
/2/     L, /4/
        *, /5/
        ST, /6/
        B, $/3/
```

A single generator for all four SMAC statements could be written in the following manner:

```
                MBGEN
                MOUTF,SK1,3,SK2,0,SK3,5
                MEXIT
SK1             MBSKL
/2/     /       L,/4/
                *,/5/
                ST,/6/
                B,$/3/
SK2             MBSKL
/2/             B,$SUBRT
                  ,/3/
SK3             MBSKL
/2/             L,/4/
                +,/5/
                -,/6/
                *,/7/
                ST,/8/
                MEGEN
```

This generator will put out SK1 if the SMAC statement has three arguments, SK2 if the SMAC statement has no arguments, and SK3 if the SMAC statement has five arguments.

11. MOUTV,M,NAME,K
    (OUTput Variable number of parameters)

This statement will put out the skeleton indicated by NAME, M times, and shift the parameter pointer K positions each time. A pointer which points at column one is used as the initial pointer.

Example:

SMAC statement:

```
MADD,A,B,C,D,E,F,G
```

This statement is to load A, add B, C, D, E, and F and store the result in G. Its generator could be written as follows:

```
                MBGEN
                MOUT,QPT1,SK1
                MOUTV,5,SK2,1
                MOUT,QPT1,SK3
                MEXIT
SK1             MBSKL
/2/             L,/4/
SK2             MBSKL
                +,/5/
SK3             MBSKL
                ST,/10/
                MEGEN
```

However, the MOUTV statement can also be written with several sets of arguments M, NAME, K as follows:

$$\text{MOUTV}, M_1, \text{NAME}_1, K_1, M_2, \text{NAME}_2, K_2, \ldots, M_n, \text{NAME}_n, K_n$$

The pointer for the first set points at column one. The pointer for set i+ 1 ($i \geq 1$) is moved over L parameters from column one, where

$$L = 3 + \sum_{j=1}^{i} M_j K_j.$$

By using this facility, the above SMAC statement generator could be written as:

```
            MBGEN
            MOUTV,1,SK1,1,5,SK2,1,1,SK3,1
            MEXIT
SK1         MBSKL
/2/         L,/4/
SK2         MBSKL
            +,/1/
SK3         MBSKL
            ST,/1/
            MEGEN
```

A third version of the MOUTV statement allows one of the M's to be variable, in which case it is written as N instead of a literal. This could be used in the generator for the above SMAC statement if the number of arguments could be variable and consequently the number of + instructions to be put out is unknown until compile time. In this case, the generator would be written as follows:

```
            MBGEN
            MOUTV,1,SK1,1,N,SK2,1,1,SK3,1
            MEXIT
SK1         MBSKL
/2/         L,/4/
SK2         MBSKL
            +,/1/
SK3         MBSKL
            ST,/1/
            MEGEN
```

The SMAC statement for this generator could be

```
MADD,A,B,C,D
```

and the output would be

```
L,A
+,B
+,C
ST,D
```

The SMAC statement could also be

MADD, A, B

in which case the output would be

```
L, A
ST, B
```

In other words, N is computed by the MOUTV statement according to the following equation:

$$N = \left[(\text{number of parameters} - 3) - \left(\sum_{j=1}^{i-1} M_j K_j + \sum_{j=i+1}^{P} M_j K_j \right) \right] / K_i$$

where $M_i = N$.

The following example demonstrates this statement in another generator.

Example:

SMAC statement

General form: TAG MADMY, arg_1, arg_2, . . . , arg_n

Definition: The first argument is loaded, the first member of each pair from arg_2 to arg_{n-2} is added, the second is multiplied, and arg_{n-1} and arg_n are stored.

Specific use:

example 1: NAME 1 MADMY, A, B, C, D, E, F, G, H, I

example 2: NAME 2 MADMY, A, B, C, D, E, F, G, H, I, J, K

example 3: NAME 3 MADMY, A, B, C

Generator:

```
          MBGEN
          MOUTV, 1, SKEL1, 1, N, SKEL2, 2, 2, SKEL3, 1
          MEXIT
SKEL1     MBSKL
/2/       L, /4/
SKEL2     MBSKL
          +, /1/
          *, /2/
SKEL3     MBSKL
          ST, /1/
          MEGEN
```

Output: example 1. N = 3 example 2. N = 4 example 3. N = 0

```
NAME  L, A        L, A        L, A
      +, B        +, B        ST, B
      *, C        *, C        ST, C
      +, D        +, D
      *, E        *, E
      +, F        +, F
      *, G        *, G
      ST, H       +, H
      ST, I       *, I
                  ST, J
                  ST, K
```

Writing SMAC Generators

The following sections contain the rules to be followed and methods to be used in writing SMAC generators.

Order of Statements in Generators

All generators must begin with the MBGEN statement and end with the MEGEN statement. In the logic portion of a generator, the generator programmer cannot use the STRAP SLC and END pseudo-ops. Except for this restriction, STRAP and SMAC statements may be freely intermixed.

The statement which follows the MBGEN statement will be the first statement of a generator to be executed. The last statement of the logic portion to be executed must be the MEXIT statement, which returns control to the processor. In the skeleton portion, each skeleton is preceded by the MBSKL statement and followed by either another MBSKL statement or the MEGEN statement. The MEGEN statement is physically the last statement of a generator.

Skeletons

A skeleton is written as a sequence of STRAP and SMAC statements, in accordance with the SMAC language restrictions specified above.

At generator execution time, the parameters of a SMAC statement are inserted into the skeleton statements of a generator exactly as they appear in the SMAC statement. The field in which a parameter is to be inserted is indicated by use of the symbol /n/, where n is the number of a parameter in the SMAC statement to be inserted. Tags for statements within skeletons can be made unique by using /Q/ in the tag. When a generator is executed, a unique tag is inserted wherever /Q/ occurs in any skeletons contained in this generator. The programmer may vary the generated tags by preceding, surrounding, or following /Q/ by a maximum of any three alphanumeric characters except Q. To insert a / into the card image, a // is used in the skeleton.

Use of Index Registers in Generators

1. Index registers are never used in Generator Language statements.

2. All references made by STRAP statements in the logic portion of a generator to statements within that generator must be modified by index register one, which allows for generator relocation.

3. Any references made by STRAP statements in a generator to routines, buffers, or control information outside the generator must be modified by index register two, which provides for linkage through the branch table.

4. Index register fifteen is used for subroutine linkage.

5. Index registers 3, 4, 5, 6, 7 and 8 are reserved for use in the skeletons of the generators for the Generator Language statements.

6. Index registers 9, 10, 11, 12, 13 and 14 may be used freely in STRAP statements in the logic portion of a generator.

Use of Branch Table

Many of the branch table entries are useful to the generator writer. Therefore, this section contains a list of these entries and their functions.

ENTRY	FUNCTION
QDBUGB	A bit which is set to 1 when processor is running in the debug mode.
QGUB	Generator usage bit. This bit equal to 0 will indicate that a particular generator is being executed for the first time during a compilation. The bit equal to 1 indicates that the generator has been executed previously.
QMIBIT	Set to one if previous processed statement was a macro. It remains set to one until first non-continuation card is processed by statement identified.
QCIBIT	If set to one, previous source statement was a macro.
QMACBIT	If set to one, macro is being expanded.
QINBIT	If set to one, inner macro is being expanded.
QERRBIT	If set to one, a major error has occurred and compilation will be terminated at the end of this pass.

IBM
International Business Machines Corporation
Data Processing Division, 112 East Post Road, White Plains, N. Y.

Printed in U. S. A. J28-6128 6/61