X-565-67-483

NASA TM X-55970

A DECOMMUTATION COMPILER

OCTOBER 1967

NASA

GODDARD SPACE FLIGHT CENTER

GREENBELT, MARYLAND

X-565-67-483

A DECOMMUTATION COMPILER

E. I. Grunby
Information Processing Division

October 1967

GODDARD SPACE FLIGHT CENTER
Greenbelt, Maryland

A DECOMMUTATION COMPILER

INTRODUCTION

The steps in performing decommutation for an experiment are comparatively few:

- Read an input tape
- Declare whether the main-frame matrix is stored in memory by row or column
- Specify what fill (or padding) characters are to be used
- Decommutate over a fixed list of row indices while the column indices vary
- Decommutate over a fixed list of column indices while the row indices vary
- Decommutate an arbitrary number of "random" elements
- Define a subroutine and its exit
- Call a subroutine
- Write the decommutated information accumulated for a given experiment on a given output tape

The function of the decommutation (decom) compiler is to generate machine-language instructions to accomplish these tasks and to check for errors in the compiler language.

Judicious use of the SLEUTH II assembly system makes it possible to define the entire set of decom compiler instructions by procedures which are available at compilation time. When a program calls the procedures and specifies the proper parameters, machine-language code is generated. Great care has been taken to detect parameter errors and to place appropriate flags on the program listing.

FORMATS

The following paragraphs describe the instruction formats for the decom compiler, with their uses, as well as details of (internal) compiler logic.

ARRAY

Format—

```
YYYYYY      ARRAY  Δ   R, C  Δ  'XXXXXX'
```

where R is the number of rows in the matrix,
C is the number of columns in the matrix,
XXXXXX is either BY ROW or BY COL for the storage method,
YYYYYY is arbitrary and may be omitted.

Use—This procedure must be used as the first instruction of an independent decom program or subroutine in order to establish core-storage information for the compiler instructions DCOMCR, DCOMRC, ELMENT. It defines the matrix dimensions and the form of storage used.

Compiler logic—The procedure ARRAY defines two absolute global labels (that is, values outside the procedure): ROW and COL, which have values of R and C, respectively. References to these labels in the main program (or other decom language procedures) will cause generation of the value at the reference point.

An additional absolute global label defined in this procedure, SBRC, will have the value (in Fieldata characters) of 'XXXXXX' as defined above.

FILL

Format—

```
XXXXXX      FILL      e
```

where XXXXXX is the label to be equated to the fill value,
e is any SLEUTH II expression.

Use—This procedure, which defines a fill or padding value, must be used before any references to it by other decom instructions. Any number of fill values may be defined; however, each label may be used only once in a given subprogram.

Compiler Logic—A check is first made to see if e is negative. If e is positive, XXXXXX is equated to -e. If e is negative, XXXXXX is equated to the complement of bits 34 = 00 of e. (The original sign bit is masked out.) An NOP instruction is generated with an operand selected to force a truncation flag on the SLEUTH II listing.

Because the foregoing label was defined at the main program level, it is available to all decom language instructions. It is global in the same sense that ROW was in the instruction ARRAY. Negative values are defined for flagging purposes, so that fill-data labels may be distinguished from matrix-element subscripts.

CALL

Format—

Δ ←→ Δ CALL XXXXXX

where XXXXXX is the label defining a subroutine declared with a SUBROU instruction.

Use—This procedure transfers to the required subroutine.

Computer Logic—Matching instruction SLJ XXXXXX is generated.

SUBROU

Format—

XXXXXX SUBROU

where XXXXXX is the label defining the entrance line of this subroutine.

Use—This procedure provides an entry point into instructions for a decom language subroutine. If the subroutine is independently assembled, XXXXXX must be externally defined by an asterisk. Subroutines may be defined at any point permissible by program logic.

Compiler Logic—Machine instruction XXXXXX J $ is generated to accommodate the entry of an SLJ call.

EXIT

Format—

Δ ⟷ Δ EXIT XXXXXX

where XXXXXX is the label on the SUBROU line defining the subroutine.

Use—This procedure is an exit line from subroutine XXXXXX. Any number of these instructions may be used; their positions within the subroutine may be arbitrary.

Compiler Logic—Machine instruction J XXXXXX is generated.

ELMENT

Format—

Δ⟷Δ ELMENT Δ p(1) Δ p(2) Δ . . . Δp(n)

where p(n) may have two forms: form r, c which uses row and column indices of a desired matrix element from the input array, or form XXXXXX which uses the label XXXXXX of a FILL instruction. Any number of parameters may be specified and continuation cards may be used.

Use—This procedure selects from left to right the indicated matrix elements from the current input matrix or fill values and stores the sequence in an output area.

This description assumes that a definition for ARRAY was made before the call of ELMENT and that, if a fill value is used, the associated label was defined earlier in the coding.

Compiler Logic—The first step is to generate a code which initializes an index register with the relative address of the next available cell in the output-storage area. The address value, labeled N, is either a zero or a nonzero value from a decom statement (ELMENT, DCOMRC, or DCOMCR) generated just previously. Next, array storage is sensed by testing SBRC against BY ROW or BY COL. Matrix elements can be properly referenced by using appropriate coding during the ELMENT procedure. The generated coding is a series of load-store pairs, one pair for each list in the calling sequence. A list in the calling sequence is usually in the form r, c.

Each r and c are checked to see whether they are valid matrix indices. If they are improper, an illegal instruction is generated to provide an I flag on the printer listing. If valid, the address of the proper element in the input area is constructed and placed in the load instruction.

If a list in the calling sequence consists solely of a label previously assigned as a fill value, a literal for this value is generated in the load instruction. Because the FILL statement complements the fill definition, a negative literal value is created. When the value of the absolutely defined fill label is positive or when a list has more than two parameters, error conditions arise for which an illegal instruction with an I flag is generated.

The "store" instruction simply places the loaded data point into an output bin called OUTPUT. The previously loaded relative bin index is used for address modification and is incremented by one after the "store" is completed.

After all load-store pairs are generated, the decom compiler causes an instruction to be assembled which stores the incremented relative bin-index back to cell N, making it available for initializing storage for a decom-storage statement.

DCOMRC

Format—

. . . DCOMRC r_1, r_2, r_3 $c_1, c_2 \ldots c_n$

Parameter r_3 may be omitted, thus implying the condition $r_3 = 1$. The number of parameters in list c is arbitrary. Continuation cards may be used.

Use—This statement permits an iterative selection of matrix elements from the current input matrix. The iterative scheme is similar to the DO statement in FORTRAN. Constant r_1 is the initial value of the row index. For each column value c_n, the row index is applied and elements r_1, c_1 through r_1, c_n are selected and stored in an output area. Row increment r_3 is then applied and elements $r_1 + r_3, c_1$ through $r_1 + r_3, c_n$ are selected and stored. This process is repeated k times (k being equivalent to $r_1 + (k + 1)\ r_3 > r_2$). Therefore, the final value of row index r_3 will never be exceeded. As in FORTRAN IV, r_2 can be negative. Omission of r_3 implies an increment of one.

To select and store fill values, it is necessary only to define fill labels before using DCOMRC and to place them at the desired positions in the list of column indices.

This procedure assumes that, in addition to defining all fill values earlier in the program, the statement ARRAY was previously used to declare the size and manner of storage of the matrix.

For example, use of the following statements will store (1,5), ϕ, (1, 2), (3, 5), ϕ, (3, 2) in the output area:

```
   ARRAY      129, 135          'BY ROW'
Z  FILL       φ
   DCOMRC     1, 3, 2           5, Z, 2
```

Computer Logic—Appropriate parameters required for performing iterations over the list of column indices must be determined. The row increment, D, is taken as stated or, if the implied format is used, the value of one is assumed. Both initial and final values of the row indices (reduced by one to translate the index origins to zero) are named I and L, respectively. A repeat count, R, is then developed by determining the integer part of (L-I)/D. Because the final row index may not be an integral number of increments from the initial row index, an exact final value is computed. The exact value, F, is R*D +I.

Parameter formats are checked before the coding for DCOMRC is generated. An illegal-operation flag, I, is generated if:

1. The number of parameter lists does not equal one.

2. The computed repeat count R is negative.

3. I or F is negative.

4. The stated initial or final row index exceeds the maximum permissible row subscript.

The technique for coding DCOMRC is similar to that used for ELMENT. Switch SBRC determines whether BY ROW or BY COL matrix storage was selected, afterwhich an appropriate internal procedure produces load-store instruction pairs. If a negative column index is detected, the complement of this value becomes a literal and is used as fill data. If column indices lie beyond the maximum column index, illegal-operation flags, I's, appear on the program listing.