STRCMACS - AN EXTENSIVE SET OF MACROS TO AID IN STRUCTURED PROGRAMMING IN 360/370 ASSEMBLY LANGUAGE

C. Wrandle Barth
Goddard Space Flight Center
Greenbelt, Maryland 20771

In spite of the confusion in the past few years over the precise meaning of the term "structured programming," there is one point which seems to be gaining increasing acceptance: the intuitive concept of a "well-structured program" cannot be totally realized through blind adherence to a few mechanical rules. Anyone who has worked with languages such as BLISS or SIMPL (particularly in a student environment) will testify that elimination of the goto in these languages does not guarantee all programs to be legible and readable. Catastrophes can be constructed from the top down. A chief programmer team can still design a horse as a camel. The real lessons of software engineering are much more in the realm of attitude, approach, and emphasis than on techniques and rules.

Although this "structured programming attitude" can be applied in any program in any language, some programming environments are more hospitable than others to it. For example, languages (such as PL/I and Algol) which provide standard control structures and arbitrary nesting impose a smaller burden on the programmer when he is constructing his control flow than do languages (such as Fortran and Cobol) which are deficient in these areas.

In general, the programmer should show preference for those languages which aid rather than impede the intellectual manageability of programs. However, other real-world considerations (such as portability, efficiency, and availability) often dictate the choosing of a language which is less than ideal for reliable program development. The recent proliferation of Fortran preprocessors has been an attempt to provide a cleaner environment for applying structured programming concepts within the framework of one dirty, but widespread, language.

In the use of assembly language, one runs into much the same problem as with Fortran. From a structured programming standpoint, assembly language is a bad choice several times over. But there are times when it must be used, whether for interfaces not available in high-level languages, to take advantage of facilities not otherwise accessible, or for efficiency reasons. In these cases, it is worthwhile to employ aids to make assembly language as hospitable as possible. Macro processors, which have been almost universally available in assembly languages

(and are only now beginning to make in-roads into high-level languages) provide a mechanism for realizing such aids in assembly language.

The use of macros to provide control structures in assembly language is not new. In December of 1970, Marvin Kessler of IBM's Federal Systems Division introduced the *CONCEPT* 14 macros. The package provided looping (do-while, do-until, indexed, and search [two exit] loops) and conditional selection (if-then-else and integer-case). Conditional tests were stated as single instructions and condition code tests, with limited capabilities for producing compound conditionals. *CONCEPT* 14 has been distributed informally by IBM FSD for some time.

During 1973, I developed a similar set of structured programming macros. Although the initial development was independent of other efforts, I incorporated ideas from other packages as they became known to me. This macro package, called STRCMACS, was first distributed in January of 1974, and remains, to the best of my knowledge, the most extensive macro package for structured programming in 360/370 assembly language.*

Since that time, STRCMACS has been distributed to about sixty installations. User reaction has been favorable. The macros have proven reliable (only five bugs were discovered, all fairly minor). This was due at least in part to the use of structured programming techniques and attitudes in their development.

I will not go into great detail on the macros' form and capabilities here since this information is available in a separate publication. The examples shown below should provide the flavor of the package.

The if-then-else and do-while provide the basic starting point:

```
IF  (LTR,3,3,Z)          DO WHILE,(LTR,3,3,Z)
  A                        C
ELSE                     OD
  B
FI
```

*There exist some full preprocessors with more extensive capabilities, but these amount to full languages which translate to assembly language.

A is executed if register 3 contains zero; else *B*. *C* is executed zero or more times as long as register 3 remains zero. Post-loop testing is provided by the UNTIL keyword:

```
DO   UNTIL,<,(CR,7,5,EQ),OR,(SR,3,1,Z),>,AND,(LRT,1,1,MASK=8)
   D
OD
```

This example also shows a complex conditional with parenthesizing via angle brackets. Several forms of case statements are provided.

Integer:

```
DOCASE   I
   CASE   3          (This block executed if I contains 3)
     A
   ESAC
   CASE   7,(9,12)   (Seven and nine through twelve)
     B
   ESAC
   CASE   MISC       (All others)
     C
   ESAC
ESACOD
```

Character String:

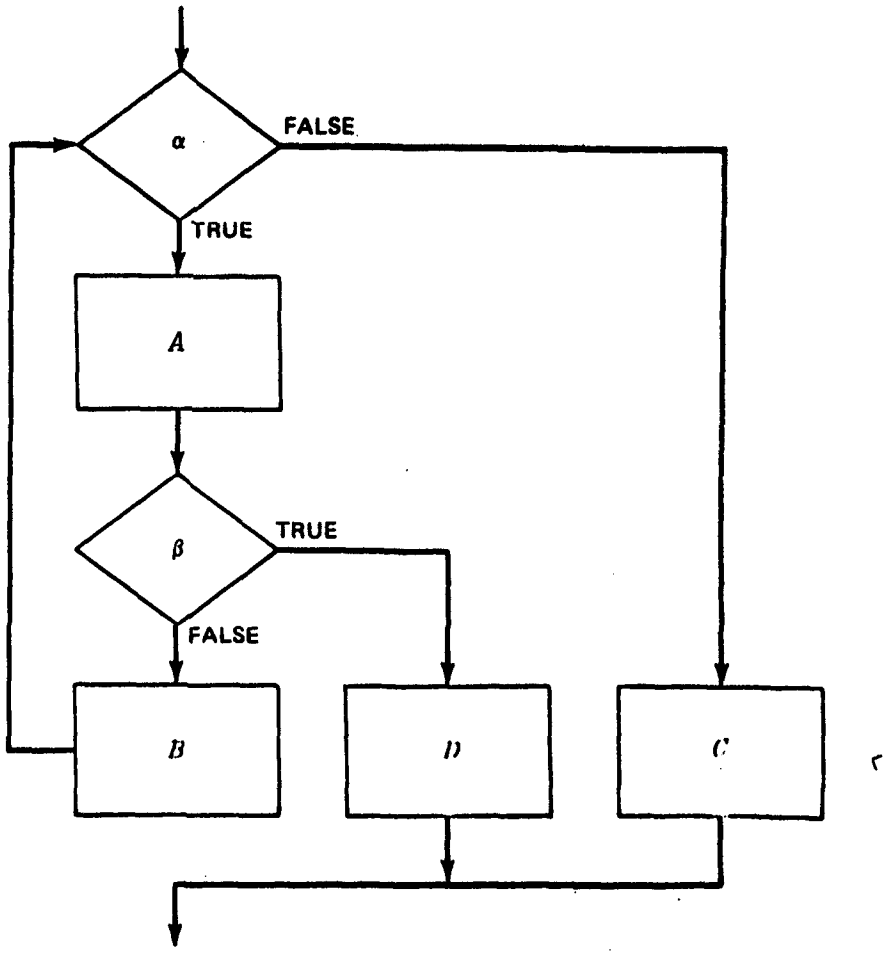
```
DOCASE   (OPCODE,4)
   CASE      'ADD '
     D
   ESAC
   CASE      'REPL','CHNG'
     E
   ESAC
   CASE      'DELT'
     F
   ESAC
ESACOD
```

Conditional Test:

```
DOCASE
   CASE   (LTR,3,3,Z)    (First CASE which evaluates true
    G                     is executed.)
   ESAC
   CASE   (CR,1,2,EQ),OR,(TM,FLAG,X'80',0)
    H
   ESAC
   CASE   (S,5,WORD,P)
    I
   ESAC
ESACOD
```

A two-exit loop capability (useful in table-searching) is available. The syntax and equivalent flow chart is shown below.

```
LOOP DO WHILE, α
    A
    IF β, EXIT = LOOP
    B
ATEND
    C
ONEXIT
    D
OD
```



In addition to control structures, the ability to create simple local procedures is useful in the structured programming environment. The PROC and CORP macros provide this capability and, in addition, handle a number of details dealing with register saving and restoring, OS and DOS linkage conventions, base register handling, and debug aids.

The above examples are far from complete. Full documentation is provided in a manual called "STRCMACS: An Extensive Set of Macros for Structured Programming in OS/360 Assembly Language."* This document is available from the sources shown below.

While STRCMACS will not turn straw programmers into gold, it is a tool which provides a framework within which the techniques and attitudes of structured programming can be more confortably exercised.

Number of macros: 34

Number of card images: about 3000

Operating System: DOS or OS

Assembler: DOS-D or F; OS-F, G, or H

Available from:

Author: C. Wrandle Barth
Code 603
Goddard Space Flight Center
Greenbelt, Maryland 20771

(Please provide minireel)

or COSMIC: University of Georgia
Suite 112, Barrow Hall
Athens, Georgia 30602

(Program No. GSC 11938)

*The most recent version of STRCMACS is DOS-compatible.