Portability of the MBASIC Machine-Independent Design

M. C. Riggins
DSN Data Systems Development Section

Part of the current work of the DSN Software Standards Project concerns the machine-independent design of the MBASIC processor. This article describes a study effort toward a portable implementation of the machine-independent design. The method made use of the STAGE2 portable, general-purpose macro processor, by means of which it was possible to invent a set of seemingly machine-independent macro templates for translation into an arbitrary target assembly language. The conclusions reached by this study are that the macros defined seem to form an adequate program MBASIC processor design language, that it is possible to carry structured programming concepts to the assembly language level, and that implementation by machine-independent macros may not be quite as efficient as hand coding but may yield significantly lower implementation costs.

I. Introduction

This article describes one phase of work done in support of the MBASIC machine-independent design (MID) activity. The purpose of the work was threefold: (1) to validate the MID design, i.e., to verify that the algorithms supplied actually perform as required by the MBASIC Language Specification (Ref. 1); (2) to evaluate whether the design supplied was truly machine-independent; and (3) to investigate methods which promote an orderly and rapid implementation of this design on an arbitrary target computer.

The design criteria specify that the design should be free of characteristics peculiar to specific host computers but should make maximum use of characteristics shared among a defined class of computers. Considerations beyond the set of common basic requirements define the MBASIC environmental interface, which is then to be undertaken as a separate implementation activity for each host computer. Thus, the implementation of an MBASIC processor into a given host computer consists of coding the MID and designing and coding the environmental routines which interface the MID to the system.

If the MID could meet its goals, it seemed reasonable that there would also be a method whereby the implementation of the MID could take place as rapidly and be as error-free as possible. That is, it seemed reasonable to seek a way of coding the MID in a machine-independent language—or at least in a language which requires only relatively minor alteration to transfer that body of code from one machine to another. Such a philosophy severely limits the choice of a programming language, for very few languages are machine-independent, widely available, and suitable for coding language processors. Languages such as FORTRAN and COBOL are largely machine-independent and widely available but inadequate for language implementation. Assembly language makes for efficient programs but is certainly not portable.

II. Macro Languages

The best choice for implementing the MID seemed to be one which would permit coding in a set of machine-independent macros, which could then be translated into an arbitrary target assembly language by merely redefining the macro definition body to fit that host system. This choice presumed that the macros themselves could take a portable form acceptable to various hosts.

Fortunately, STAGE2 (Ref. 2) seemed ideally suited for this purpose. STAGE2 is a general-purpose, readily available, portable macro generator/processor that can be installed in any computer capable of accepting ANSI-FORTRAN. STAGE2 permits its user to define macro "templates", which, when sensed in the source stream, then cause the generation of assembly language code also supplied by the user. Moreover, STAGE2 was already available on the U1108, so no further effort was required to begin in the MBASIC portability study.

Furthermore, even if STAGE2 were not available on another particular computer, the macro instructions, with proper specifications, checked and certified correct by previous implementation on the U1108, could serve as the basis for coding on the envisioned host. In such a case, the programmer himself is a manual macro translator. Further, since the U1108 implementation could be certified correct, any failures in other implementations could be isolated to coding errors on the new host system.

III. Portability of MBASIC

The envisioned process for implementing MBASIC into a given system is illustrated in Fig. 1. The figure shows several levels of documentation and their interrelationships. The MBASIC Language Specification is contained in a set of manuals (Ref. 1), which lead to the MID as a set of flowcharts and environmental interface specifications (Ref. 3). Upon study of the flowcharts, a set of macro specifications can be generated sufficient to encode the MID. An outline of such specifications appears in Appendix A of this article. The specifications, along with the assembly language manuals for the host, lead the programmer to providing proper macro body definitions in the host assembly language. Then, the MID-macro instructions can be translated (either by STAGE2 or manually) into a host assembly language implementation of the MID.

The machine-dependent design (MDD) proceeds similarly, building the capability required by the environmental interface. The figure shows that some of the MID macros may be useful in the MDD and should, of course, be available to the MDD programmer, as well as the assembly language of the host. The resultant MDD assembly language, together with that for the MID, forms the full MBASIC processor.

IV. The MID Macro Set

The MBASIC-MID is contained in a set of flowcharts which follow a structured programming topology. For this reason, a set of CRISP (Ref. 4) control structures, such as IF...THEN...ELSE, LOOP...EXIT...REPEAT, and DO CASE, plus the MBASIC design conventions were determined to be sufficient for the purpose of coding the MID. To implement these macros, a set of *utility macros* was designed and coded for stack manipulation, label generation, and flow-of-control, all of which are for internal use of the macros used to code the MID. The entire set of macros has four subdivisions:

(1) Machine-independent system utility macros

(2) Machine-dependent utility macros

(3) Machine-independent MID macros

(4) Machine-dependent MID macros

These subdivisions are described below:

(1) Machine-independent system utility macros—These macros function as routines to be used by other macros for tasks such as stack manipulation, label generation, and comments.

(2) Machine-dependent utility macros—These routines generate assembly language code for addressing, branching (conditional and nonconditional), subroutine linkage, and carriage control. This set, like the

system utility macros, is for internal use by other macros and will not appear in the code for the MID.

(3) Machine-independent MID macros—These macros are the flow-of-control macros. They do not generate any assembler code directly. Indirectly, through calls to other macros, they generate code and manipulate labels for the control structures IF...THEN ...ELSE, UNLESS...THEN...ELSE, DO CASE, and LOOP...EXIT...REPEAT.

(4) Machine-dependent MID macros—These macros generate code directly and handle procedure and subroutine linkage, variable declaration, arithmetic operations, and MBASIC design conventions.

There are two basic lines of division within this set of macros: independence/dependence and utility/non-utility. Independence/dependence separates the macros that generate assembler code (dependence) directly from those that do not (independence). Utility/non-utility separates the macros that appear in the MID design code (non-utility) from those that do not (utility). Using these macros, the MBASIC MID could be coded using the same structures that the flowcharts were written in. Verification of the translation from flowchart to code was indeed made much easier because of this. Additionally, each non-utility macro generates an assembly code comment that aids in understanding and debugging the assembly code, because it specifies which macro call generated the code that follows that comment.

As the design and implementation of the set of macros proceeded, checking the macros, individually and in groups, was also included. After a preliminary set of macros, as defined by the specifications derived from the MID (except for a few of the MID conventions), was designed, implemented, and tested, a test program was written to exercise all possible macros, directly or indirectly. The test program is shown as Appendix A. Even without a detailed macro specification, the test procedure is fairly readable and understandable—far more so than if the macros were more assembly-language-like.

This test program and the macros were run through the STAGE2 system; the code was generated, assembled, and executed. The output from this run, shown as Appendix B, was verified to be that specified by the design of the test program.

V. Coding the Design

The coding of the MBASIC MID was to take place in multiple phases. However, at this writing, none of these has begun. The first phase was to involve coding and testing the first three tiers of the MID using dummy stubs (Ref. 5) in place of references to modules at tier 4. In succeeding stages, all ten tiers of the MID were to be implemented. The implementation was not necessarily meant to compete with the current operational MBASIC processor on the 1108 in terms of efficiency. The code generated by STAGE2 would, hopefully, be equivalent to the operational processor, but it would probably be somewhat slower because of the limited optimization capability of STAGE2. This was considered to be of minor importance initially, because our aim was to implement the design in a portable fashion even if it were likely to be less efficient than coding the design by hand for a particular machine.

When that design had been macro-coded and the macro-coding verified, it would form a correct, compilable source for multi-implementations. Hand coding or machine translation and optimization could conceivably then make the operation very efficient, and more importantly, could proceed, with the knowledge that any errors detected were not in the design but somewhere in the coding.

VI. Conclusions

The proof of the method described is incomplete, and awaits an actual implementation for validation of the techniques proposed. However, one conclusion can be drawn unequivocally at this point: Carrying top-down structured programming concepts to the assembly language level for a particular computer needs no more support than a macro processor such as STAGE2. Less strongly, but with some degree of assurance, it seems fair to state that the set of macros specified for the complete MID form an adequate program design language (PDL) for the MID which will serve as a better, more definite basis for implementation than do the flowcharts. As a final conclusion, it also seems fair to state, based on the experience here, that portable machine-independent portions of systems appear feasible and desirable but may lead to some lack of efficiency in execution speed and core utilization; however, they may yield significantly lower initial implementation costs.

References

1. *Fundamentals of MBASIC*, Vols. I and II, Jet Propulsion Laboratory, Pasadena, California, March and October 1973 (JPL internal document).
2. Waite, W., *Implementing Software for Nonnumeric Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.
3. *Software Specification Document: Machine Independent MBASIC*, Jet Propulsion Laboratory, Pasadena, California, to be published (JPL internal document).
4. Tausworthe, R. C., "Control-Restrictive Instructions for Structured Programming (CRISP)," *The Deep Space Network Progress Report 42-22*, pp. 134–151, Jet Propulsion Laboratory, Pasadena, California, 1974.
5. Baker, F. T., *Chief Programmer Teams: Principles and Procedures*, Report FSC 71-5108, IBM Federal Systems Division, Gaithersburg, Maryland, June 1971.

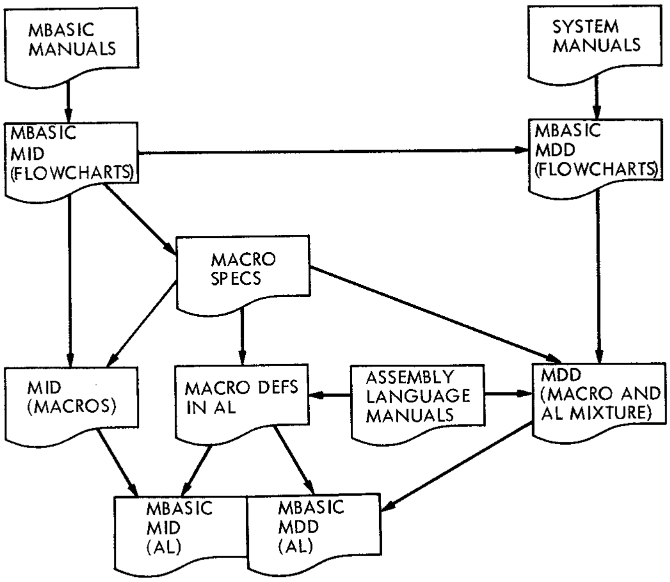

Fig. 1. Implementation of the MBASIC processor by machine-independent macro structures

Appendix A

```
:PROGRAM: MACTST                                                        MOD# 1;
.1   :DECLARE Y1(1);  DECLARE AND INITIALIZE
     :DECLARE Y2(1);  LOOP INDEX VARIABLES
     :DECLARE Y3(1);
     :DECLARE Y4(1);
     :DECLARE DX(1);
     :Y1=(3); ASSIGN POSITIVE LITERAL VALUE
     :Y2=(1);
     :Y3=Y2; ASSIGN VALUE OF VARIABLE (=1)
     :Y4=(3);
     :DX=(-1); ASSIGN NEGATIVE LITERAL VALUE
     :PRINT THE FOLLOWING PRINTOUT SHOULD APPEAR IN NUMERIC ORDER;
.3   :LOOP :DO CASE Y1 OF 1 TO 3;
.4           :CASE 1::PRINT 3. FINAL CASE PRINTS LINES 4-10;
.5                 :DO PROCED; TEST DO,CALL,IF,LOOP,REPEAT
                   :END;
.6           :CASE 2::PRINT 2. UNTIL LOOP AND ADDITION MACRO;
                   :END;
.7           :CASE 3::PRINT 1. FIRST DO-CASE TRANSFER;
                   :ENDBLOCK;
.8       :Y1=Y1 + DX; ADDITION OF VARIABLE VALUE (=-1)
       :EXIT IF (0) >= Y1;
         :REPEAT;
.9   :PRINT MACRO TEST TERMINATED;
     :ENDPROG;
```

```
:PROCEDURE: PROCED                                                      MOD# 1.5;
<*  TESTS CALL,IF,UNLESS,LOOP,AND REPEAT MACROS,AND *>;
<*  ALSO INCLUDED UTILITY MACROS UNDER =,#,<,>,>=,<= *>;
.1 :LOOP :EXIT UNLESS Y2 <= (7);
.2     :IF Y2 < (4);
.3       :THEN :IF Y4 > Y2;  TEST VARIABLE VALUES (Y4=2).
.4               :THEN :IF Y2 = (1);
.5                       :THEN :PRINT 4.  TEST VALUES WITH <,>,=;
                               :END;
.6                       :ELSE :PRINT 5.  TEST VALUES WITH <,>,#;
                               :ENDBLOCK;
                     :END;
.7               :ELSE :LOOP :PRINT 6. THIS LINE WILL BE PRINTED 3 TIMES;
.8                         :Y3=Y3 + (1);
                       :EXIT UNLESS Y3 <= (3);
                         :REPEAT;
                     :ENDBLOCK;
             :END;
.9/S1    :ELSE :CALL SUBRTN; TEST CALL,UNLESS,REPEAT
             :ENDBLOCK;
.10      :Y2=Y2 + (1);
         :REPEAT;
    :ENDPROC PROCED;
```

```
:SUBROUTINE: SUBRTN                                                  MOD# S1;
<* TEST SUBROUTINE NESTING, UNLESS  ,REPEAT *>;
.1  :UNLESS (5) < Y2;
.2    :THEN :UNLESS Y2 # (4);
.3              :THEN :PRINT 7. Y2 <= 5 AND Y2 = 4;
                      :END;
.4              :ELSE :PRINT 8. Y2 <= 5 AND Y2 # 4;
.5                    :UNLESS Y2 = (5);
.6                      :THEN :PRINT ***ERROR IN UNLESS Y2=(5);
                              :END,NOELSE;
                      :ENDBLOCK;
            :END;
.7/S2 :ELSE :CALL SUBSUB; TEST NESTING, REPEAT
             :ENDBLOCK;
    :RETURN;


:SUBROUTINE: SUBSUB                                                  MOD# S2;
<*   TEST SUBROUTINE NESTING, REMAINING UNLESS, AND REPEAT *>;
.1    :UNLESS Y2 > (6);
.2      :THEN :PRINT 9. Y2 <= 6;
.3            :IF Y2 # (6);
.4              :THEN :PRINT  ***ERROR IN IF Y2#(6);
                      :END,NOELSE;
            :END;
.5/S3  :ELSE :CALL SBSBSB; TEST NESTING, REPEAT
             :ENDBLOCK;
      :RETURN;


:SUBROUTINE: SBSBSB                                                  MOD# S3;
<*    TEST NESTING, REPEAT  *>;
.1    :LOOP :PRINT 10. THIS LINE WILL BE PRINTED TWICE;
.3        :Y4=Y4 + (-1); ADDITION OF NEGATIVE LITERAL
.4/S4     :CALL SSSS(Y4); TRUTH VALUES + ARGUEMENT PASSING
        :EXIT IF (1) >= Y4;
          :REPEAT;
      :RETURN;


:SUBROUTINE: SSSS                                                    MOD# S4;
.1    :A1=TRUE(#1 = (2)); Y4 IS PASSED PARAMETER
.2    :IF A1 IS TRUE;
.3      :THEN :PRINT 11. Y4 = 2  UPON ENTRY TO SSSS;
              :END;
.4      :ELSE :PRINT 12. THIS PATH TAKEN ON SECOND CALL TO SSSS;
              :ENDBLOCK;
      :RETURN;
$FINISHED;
```

Appendix B

```
THE FOLLOWING PRINTOUT SHOULD APPEAR IN NUMERIC ORDER
1. FIRST DO-CASE TRANSFER
2. UNTIL LOOP AND ADDITION MACRO
3. FINAL CASE PRINTS LINES 4-10
4.  TEST VALUES WITH <,>,=
5.  TEST VALUES WITH <,>,#
6. THIS LINE WILL BE PRINTED 3 TIMES
6. THIS LINE WILL BE PRINTED 3 TIMES
6. THIS LINE WILL BE PRINTED 3 TIMES
7. Y2 <= 5 AND Y2 = 4
8. Y2 <= 5 AND Y2 # 4
9. Y2 <= 6
10. THIS LINE WILL BE PRINTED TWICE
11. Y4 = 2  UPON ENTRY TO SSSS
10. THIS LINE WILL BE PRINTED TWICE
12. THIS PATH TAKEN ON SECOND CALL TO SSSS
MACRO TEST TERMINATED
```