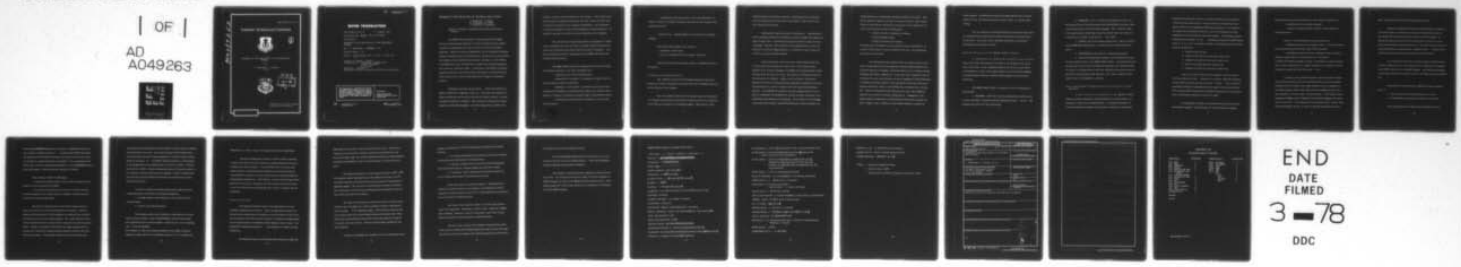
AD-A049 263 FOREIGN TECHNOLOGY DIV WRIGHT-PATTERSON AFB OHIO F/G 9/2
PROBLEMS IN THE REALIZATION OF THE MERA-BASIC SYSTEM,(U)
AUG 77 J BANKOWSKI, J DOBOSZ, K FIALKOWSKI

UNCLASSIFIED FTD-ID(RS)T-1247-77 NL

| OF |
AD
A049263



END
DATE
FILMED
3-78
DDC

AD A049263

FTD-ID(RS)T-1247-77

FOREIGN TECHNOLOGY DIVISION

PROBLEMS IN THE REALIZATION OF THE MERA-BASIC SYSTEM

by

J. Bankowski, J. Dobosz,
et al.

Approved for public release;
distribution unlimited.

EDITED TRANSLATION

FTD-ID(RS)T-1247-77 17 August 1977

MICROFICHE NR: FTD-77-C-001057

CSI76286783

PROBLEMS IN THE REALIZATION OF THE MERA-BASIC SYSTEM

By: J. Bankowski, J. Dobosz, et al.

English pages: 19

Source: Informatyka, Vol. 11, Nr. 4, 1976, PP. 2-5

Country of origin: Poland
Translated by: LINGUISTIC SYSTEMS, INC.
F33657-76-D-0389
F. Zaleski
Requester: FTD/ETCK
Approved for public release; distribution unlimited

PROBLEMS IN THE REALIZATION OF THE MERA-BASIC SYSTEM

J. Bankowski, J. Dobosz
K. Fialkowski, M. Halski
T. Sarnecki, B. Szymanski

Institute of Scientific-Technical and Economics Information,
Warsaw

The MERA 300 minicomputer series produced by the MERA Institutes of Minicumputer Systems is a family of small electronic digital computers arousing wider and wider interest of the users of computer equipment. Offered in various configurations, with a continually increasing number of available types of external devices, they find use in various fields. The software of these machines has also been developed. In the software of the MERA 300 series the department of numerical calculation programs was to this time relatively weaker. In particular, elaboration of programs appeared necessary, which would enable calculations to be performed with rather great accuracy and for large capacities of change of represented numbers.

Belonging to the basic factors which decide the suitability of digital computers for consumers' needs are: convenience and simplicity of system maintenance and the size of actual cost of labor essential to solving calculation problems by computer. The necessity of writing and actuating programs in assembly languages, even when using packs of standard sub-

routines, is often a serious problem for the consumer. This concerns particularly those calculations executed not very often, where the effort used to prepare the program can be completely unprofitable. For convenience in setting the program in motion the following have a decided influence: the manner of input and correction as well as responsible error diagnosis.

The realization of programming systems in languages of a higher order, including at the same time built-in suitable editorial functions and procedures facilitating conversation with the system is meaningful from these viewpoints for offered configurations of the MERA 300 system and available external devices.

The MERA-BASIC system was designed with the idea of solving the problems mentioned. The basic foredesigns were:

-- construction of a conversational system

-- realization of a translator of a language of a higher order for use in scientific-technical calculations

-- building in, in the system, an arithmetic block and routine functions for floating-point calculations with claims by the computer manufacturer: accuracy of calculations and great range of represented numbers

-- providing the system with great resistance to errors made by the consumer

-- resolving the processes of input routine and translation in a manner to allow for avoiding troublesome operations for the consumer with an external carrier.

A subset of the language BASIC was accepted as the program language.

The system was designed in two versions:

-- simplified, without disks

-- full, for configurations with a magnetic disk store

The article discusses the already realized, simplified version of the system.

Problems of Implementation

The realization of the brief foredesigns presented required the solution of a series of problems arising both from the foredesigns themselves and the features of the computer.

The conversational character of the system requires the possibility of frequent interlacing of the processes of input routine and its translation, correcting, and carrying out a translated program. With respect to this,

individual parts of the system's software, realizing the above mentioned activities during the entire work time of the system, must be found in the store with little access time.

Attaining the required accuracy of calculations demands provision for keeping variables and constants of a useful program with sufficiently large storage areas. Calculations with great accuracy are also very time-consuming. Likewise, other qualities of the designed system, such as resistance to errors and diagnosing them, or simplicity in input routine and correction, increase storage demands.

On the other hand, in the case of the version without disks only 8 k 8-bit words of a working store can be used. At the same time the qualities of the computer language of the MERA 300 computers effect further increase of the store and time loads. The absence of arithmetical instructions (except for the simplest: adding to the accumulator, subtraction, shifting 1 bit) forces to the programming organization arithmetical functions complicated still by a short computer word with long representations of numbers. In the MERA 300 computers the only available specific form of indirect addressing--automodification, also has an effect sometimes on the complication and extension of the program. As is evident, the foredesigns of the system and computer characteristics present critical demands on an

indispensable store and attainable operational speed of the system. With these conditions it became necessary to accept a series of rather drastic limits in order that the realization of the project could be possible at all. Limits were introduced in the following areas

-- number and kind of represented variables

-- realized language subset

-- auxiliary functions of the system.

Postulates were maintained to ensure great accuracy of calculations, resistance of the system to errors committed by the user, and adjustments in the area of corrections.

The limitations of the realized subset were input in such a way to reach a compromise between indispensable programming possibilities and desirable elasticity of language, and the dimensions of an accessible storage. (Language description--appendix A). At the same time resignation from the realization of a loop and subroutines as well as assignment of fixed addresses to variable ones makes possible a full translation of each line of the source program separately, which in a remarkable way streamlines the conversation. Lines are translated directly when they are input, while signalled syntactic errors may be corrected during the run. Assignment in the target program of equivalents of individual lines of the source program enables a change, shift or addition of a line without having to translate the

whole program. An additional convenience during program input is the possibility of error correction during line writing or input, or cessation after writing.

For the realization of individual functions of the system algorithms are used which are characterized by small requirements concerning operational stores. On account of this, algorithm operations are sometimes knowingly given up because of speed.

Overall Structure of the MERA-BASIC System

In designing the system the premise was accepted that in view of the lack of space in the memory, the program in the source form is not stored. In the compilation phase a line of the source program is translated on a "pack" of the target program in an intermediate language. In the performance phase the combined target program is executed by an interpreter.

The MERA-BASIC system is composed of the five following functional blocks:

1. SCANNER- reads line of source program from external device to store and makes a composite analysis signalling eventual errors. The program also serves the call of directives

2. COMPILER- tests correctness of arithmetical terms, and then using the form of the lines prepared by the SCANNER, generates codes of macroinstruction of the intermediate language. The produced "pack" of the target program answering one line is provided with a line label and a numerator of computer words held by this "pack"

3. LOADER- sets up the target program from "packs" prepared by the COMPILER aligning them according to the increasing numbers of labels

4. INTERPRETER- performs the created target program

5. A block of floating-point arithmetic and of standard functions are also realized as part of the MERA-BASIC system. The program ensures the exercise of the four basic arithmetical operations and of involution with accuracy to 14 marked numbers and of the calculation with such accuracy of the value of the following routine functions: sine, cosine, absolute value, square root, natural logarithm, exponent.

Some Algorithms of Optimization in Using the Storage Element

In the course of realization of the MERA-BASIC system a series of solutions is accepted which have as their purpose the maximum economy of the storage element. An adequate distribution of functions between the compiler and interpreter seemed to be particularly

important from this point of view, as well as the adoption of "economical" solutions in these same blocks. One such solution is the adoption of non-typical sequences of translating arithmetical expressions. It is decided by the depth of the parenthetical structure; first to be translated are those fragments of the term which are placed the deepest. The following example illustrates this principle: for the arithmetical expression A*(B/C*(D-E)), where A, B, C, D, E- are simple variables, the following macroinstruction sequence is generated:

1. load D on top of the pile
2. remove E from top of pile and leave result on top
3. multiply top of pile by C and leave result on top
4. add B to top of pile and leave result on top
5. multiply top of pile by A and leave result on top.

Thus, as is evident from the above example, only the macroinstruction to load on the pile extends it by 1 element, while the arithmetical macroinstructions operate only on its top. In using such a method of translating arithmetical expressions both the length of the pile and the size of the target program answering the translated expression are much smaller than translating the expression "from left to right."

In designing the translator a key problem was the selection of an intermediate language. The advantages of a rich intermediate language

whose particular macroinstructions have a wide range of operation are:

-- simplification of the compiler program

-- creation of target programs containing a small number of macroinstructions.

However, the disadvantages of such a solution are:

--extending the interpreter program because of the large number of macroinstructions and the wide range of their operation

-- extending the instruction word, arising from the necessity of keeping a larger number of macroinstruction parameters.

A simple intermediate language is indeed an extension of a target program and a certain complication of a compiler program, but it makes for a short interpreter program and short instruction word.

A specific of the realized translator was the fact that the effects of the selected solutions were more essential on the size of the translator program than on the length of the target program (the store designed for the target program equals barely 12% of the store designed for the translator program). For this reason, and also because of the simple source language and meager target language the intermediate language was chosen also on a low level. It is composed of 31 macroinstructions: control, arithmetic (operating on the pile, at least one operand becomes the top of the

pile), calling routine functions and operations on the data.

Because the length of the instruction word should be a multiple of a bit (with respect to convenience of access to the instructions word), it was decided that codes of the operation will be of various lengths: for address macroinstructions 3-bit were mostly used, a for the rest, 8-bit. Such a solution made it possible to place in a 1-bit instruction word the most important address macroinstructions: arithmetical and calling routine functions as well as almost all non-address macroinstructions.

For a macroinstruction with a variable length a twofold macroinstruction coding was used. This principle will be explained in the example of the INPUT instruction of the BASIC language. This instruction attempts to give values input from an external device to variables found on the INPUT list.

The execution of this instruction for variables series depends, thus, on:

1. input of value from the external device to the store
2. forwarding the value under the address of the variable

The second operation is realized by the arithmetical macro-

instruction REMEMBER generated in the case of a substitution instruction. Let n signify the length of the list of variables of the INPUT instruction. The appearance of the INPUT macroinstruction forces in the control loop of the interpreter a single performance of operation 1, for n-sequential turns of this loop. After the macroinstruction INPUT, n of the REMEMBER macroinstructions appear, whose performance operation 2 realizes.

Such a solution affords two advantages:

-- shortens the target program, since it does not require n-time insertion of the macroinstruction INPUT

-- the control loop operated as if only one- or two-bit macroinstructions appeared in the target program, although in actuality macroinstructions of variable length appear.

The next two solutions whose use provided certain economy of space in the store are connected with operations on numbers. The first depends on the placement of a field designated for numbers from the DATA instruction in the area of the target program. The second uses the twofold coding of numbers from the external format--decimal to the internal format--binary. Thanks to placing the DATA field in the target program area the necessity of reserving the storage element destined for numbers from this instruction vanishes. An advantage of this solution is the fact that after

canceling the line including the instruction DATA, the place taken by numbers from this field is recovered. The concept of using twofold coding of numbers arose due to the more frequent appearance of numbers having a lesser amount of numerals. In the BASIC realized translator it is permissable for the programmer to use numbers from the interval (-10E38, -1.0E-38] and [1.0E-38, 10E38), in which the maximum amount of numerals equals 14. For numbers in binary format 8 bits are designed. Instead of keeping numbers from the instruction DATA and literals in the binary format a semi-binary format is used.

In order to estimate the storage element gain achieved in such a representation let us introduce the following designations

u - average number of bits sufficient to store numerals of the recoded number

k - percent of non-whole numbers.

The average number of bits sufficient to keep numbers in a semi-binary format-m-equals: $m=(1-k)(1+u)+k(1+u+1)$, where the first component illustrates the part of whole numbers, and the second, of the remaining ones. From this $m=1+u+k$

For example, for $k=0.5$ and linear distribution of the length of numbers ($u=4$) the average number of occupied bits is equal to 5.5. In actuality the

distribution is, rather, closer to the exponential and thus (u=2-3)m=4.

The gain averaging about 4 bits for a number makes it possible to shorten the table of literals by half and an equally important economy of storage in the case of numbers from the DATA field. Since the semi-binary format is a transitional format between the internal and external representation of numbers, the above method does not entail any additional programming complications. Unfortunately, economy of the storage element gives rise to time loss. Recoding in the semi-binary format to the binary occurs every time the executed program uses a literal or number from the DATA field.

Results Achieved

The realized translator allows for the performance of source programs containing about 100 lines. This is an approximate number and arises from the experiments of the authors, since the number of lines of the source program, which does not give occasion to crossing of storage planned for the target program to a significant extent depends on their content. The possibility of executing programs of such magnitude is viewed as being satisfactory.

The possibility exists of performing large programs through their

segmentation into sections, which are placed into the store. Because the transition from one section to another requires the introduction of a new section from paper tape, the sections should be performed in a fixed sequence and within the possibility of only one time during the run of the whole program.

The translator operates on a wide range of numbers (-10^{39}, 10^{39}). The smallest number represented by a value different from zero is 10^{-38}. Translator operation is characterized by great accuracy of calculations (14 significant digits). The cost borne in achieving this accuracy is operation time, particularly critical during recoding operations performed in the process of input and read-out of numbers.

For users not interested in such great accuracy a version is generated in which calculations are made in singular precision, assuring the user of results with 5 significant digits. This version is characterized by an 8 times faster time of performing arithmetical functions and 7 times faster time of performing the operations of input and read-out of numbers in relation to the first version. This time is particularly marked for non-whole numbers.

In order to minimalize the occupancy of the storage element in the

instance of realization of a block of floating-point arithmetic and routine functions:

1. the simplest algorithms were selected to perform the arithmetical functions and calculations of routine functions

2. a special, modular structure was used of the floating-point block and routine functions, using the appeal call of the sub-routines

3. in principle, tests to detect particular situations which can serve to hasten performance of operations are not used.

On the other hand, to increase the speed of operations and to maintain the desired accuracy of calculations a normalization is used of operands of routine functions to intervals, in which algorithms of approximations of functions are sufficiently fast.

The result of the solutions accepted is the rather long duration times of the operations. Nonetheless, thanks to this, a significant storage gain is attained. Moreover, where it was possible, these same storage areas were used for various purposes by individual blocks.

The use of both versions of the translator is particularly useful. In the course of writing and actuating programs the quick version with singular precision can then be used and the retested program can be performed

by using the version with the double precision.

The realized MERA-BASIC system can find wide use as a convenient tool for the science of programming as well as the realization of typical engineering and design calculations.

The translator described has been available for users since the end of 1975. It is foreseen that from the middle of 1976 the translator of BASIC language will work for the MERA 300 disk computer series. Developing storage with a disk storage allows for the realization of a translator of the full BASIC language.

MERA-BASIC language in expanded BNF notation

< basic signs > : := < letter> | < number > | <other sign > | |

<letter>: : = A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

<numeral> : : = 0|1|2|3|4|5|6|7|8|9

<other sign> : : = + | − | × | / | = | | < | > | (|) | , | .

<whole number> : : = [< number>] 14

<fraction>: : = · [0] [<number>] 1

<number frac.> : : = [<number>] ·[< number>]

< sign> : : = [+] | -

< index>: : = E < sign> [< number>]

<number>: : = < whole number > | < fraction>|<number frac.>|

<number> <index>

<number with sign> : : = <sign> <number>

<variable>: : = <letter>

< operator>: : =<arith. operator> | <relat. operator>

<arith. operator>: : = <add. type operator> | <mult. type operator> | +

<add. type operator>: : + | -

<mult. type operator>: : = X | /

<function name>: : = SIN|COS|EXP|ABS|LN|SQR

<functional element>: : = <function name> (<expression>)

< element>: : = <number>| <variable> | <functional element> |(expression>)

<factor>: : = element < element> XX <element>

<component> : := <factor> | <component> <add. type operator> <factor>

<expression> : := <component> | <sign> <expression> | <expression>
<mult. type operator> <component>

<instruction> : : <LET instr.> | <READ instr.> | <DATA instr.> |
<PRINT instr.> | <GOTO instr.> | <IF instr.> |
<INPUT instr.> | <RESTORE instr.> | <END instr.> |
<commentary>

<LET instr> : : = LET <variable> = <expression>

<List of variables> : : = <variable> | <list of variables> , <variable>

<READ instr.> : : = READ <list of variables>

<fixed list> : : = <number with sign> |
<list of fixed> , <number with sign>

<DATA instr.> : : DATA <list of fixed>

<list of expressions> : : = <expression> | <list of expressions> , <expression>

<PRINT instr> : : = PRINT <list of expressions>

<no. of lines> : : = [<number>]

<GOTO instr.> : : = GOTO <no. of lines>

<commentary> : : = REM [<letter> | <number> | <other sign>]

<relat. operator> : : = = | > | <> | < | = < | =

<IF instr.> : : = IF <expression> <relat. operator> <expression>
THEN <no. of lines>

<END instr.> : : = END

<RESTORE instr.> : : = RESTORE

<INPUT instr.> : : = INPUT <list of variables>

<BASIC instr.> : : = <No. of lines> <instruction>.!

<BASIC program> : : = [<BASIC instr.>]

NOTE: -- spaces are insignificant signs

-- number of lines < 4096.

-- last instruction executed by program must be instr. END

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER FTD-ID(RS)T-1247-77	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* PROBLEMS IN THE REALIZATION OF THE MERA-BASIC SYSTEM		5. TYPE OF REPORT & PERIOD COVERED Translation 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. Bankowski, J. Dobosz, et al.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Foreign Technology Division Air Force Systems Command U. S. Air Force		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE 1976 13. NUMBER OF PAGES 19
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)* Approved for public release; distribution unlimited		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*		
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)* 14		

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

DISTRIBUTION LIST

DISTRIBUTION DIRECT TO RECIPIENT

ORGANIZATION		MICROFICHE
A205	DMATC	1
A210	DMAAC	2
B344	DIA/RDS-3C	8
C043	USAMIIA	1
C509	BALLISTIC RES LABS	1
C510	AIR MOBILITY R&D LAB/FIO	1
C513	PICATINNY ARSENAL	1
C535	AVIATION SYS COMD	1
C557	USAIIC	1
C591	FSTC	5
C619	MIA REDSTONE	1
D008	NISC	1
H300	USAICE (USAREUR)	1
P005	ERDA	1
P055	CIA/CRS/ADD/SD	1
NAVORDSTA (50L)		1
NASA/KSI		1
AFIT/LD		1

ORGANIZATION		MICROFICHE
E053	AF/INAKA	1
E017	AF/ RDXTR-W	1
E404	AEDC	1
E408	AFWL	1
E410	ADTC	1
E413	ESD	2
	FTD	
	CCN	1
	ETID	3
	NIA/PHS	1
	NICD	5