Page Denied

Next 3 Page(s) In Document Denied

# AUTOMATIC CODING SYSTEM SAKO

Leon Łukaszewicz, Warszawa

## SUMMARY

Some properties are described of the automatic coding system - SAKO for medium scale digital computers XYZ and ZAM II. This system considers a range of specific features of the machines, as for instance, fixed-point arithmetic, and contains logical operations on machine words. While designing SAKO, one tried to obtain a sufficiently effective system for the elimination of machine language programming almost in the whole field of numerical and logical problems.

## INTRODUCTION

In connection with the growing need of new programs for electronic digital machines, the use of automatic coding systems is especially helpful. Their purpose is an essential shortening of time for preparing programs with the assumption that the method of solving problems is known. Out of systems used nowadays the most known are FORTRAN, MATH-MATIC and MANCHESTER AUTOCODING SYSTEM. But the use of them is still limited. For instance, in problems containing logical operations, when the need arises to use the internal structure of a word of the machine the above mentioned systems are not the most suitable. Moreover, in some systems the effectiveness of the object program is not great, and therefore, they are not economical, especially in often repeated programs. So, automatic coding systems used at present have generally a helpful character for the basic system of machine language programming.

The Automatic Coding System SAKO was conceived as a basic system which would make the programming of numerical and logical problems in the machine language rather unnecessary. Although, instructions written in SAKO may be relatively easily used jointly with machine instructions written in the System of Symbolic Addresses - SAS, this need is rather exceptional. In fact, the purpose of SAS is to be the medial step between SAKO and the real language of the machine. It is also forseen that the creation of Problem Languages will be based on SAKO.

The basic assumptions which were taken into account while designing SAKO are the following:

1. This language should take advantage of all machine possibilities and cover the full field of numerical and logical applications.
2. The resulting object program ought to be as efficient as possible concerning the time of performing it, and the occupying of place in the machine storage.
3. This language ought to be as easy as possible in the programming, and independent of special machine features.

It is easy to state, that the above mentioned points, especially the first and the third one, represent opposite tendecies. SAKO represents a certain compromise in which, however, the first two points are clearly given pririty. In spite of this, as we shall see later, SAKO is almost as general and easy in use as other above mentioned languages.

In general, it seems that the practical realization of the fully universal language, as abstracted from every individual machine feature, will always lead to compromises to the disadvantage of a fully economical use of this machine.

The fact whether the machine is fixed or [illegible]
or decimal,whether it can read magnetic [illegible]
rections,must influence the programming system [illegible]
principles given in points first and second [illegible]
of differences will always arise from the [illegible]
dencies to improve the universal language which would [illegible]

ficult to publish as the best of all possible languages [illegible]
the future.
The differences in practical languages designed for different machi-
nes should not be trated as a misfortune,if their general structure
and ressemblanse are kept.The essential matter is that the general
simplicity in using autocodes makes easier the transfer from one lan-
guage to another than when dealing with two machine languages.

To obtain similar languages built for different machines it would be
advantageous to establish some generally accepted pattern,such as
ALGOL [4] .The multiple advantages of this language were taken as
the starting point to elaborate SAKO.
In the present paper instead of a description the details of SAKO,
features are given,which distinguish it from the automatic coding
systems of common use.The description of arithmetical formulae,as
well as the way of using subroutines in SAKO are given in paper [5].

## GENERAL STRUCTURE OF SAKO. INSTRUCTION AND DECLARATION LIST.

Programs written in the SAKO language are made up of a set of senten-
ces.
We have several different types of SAKO sentences,each of which has
a definite form and meaning,and the use of each is contained in a
number of rules.Therefore SAKO is a formalised language.
The information processing performed by the machine consists of two
basic stages:

- introducing the program into the machine,
- processing the introduced program by the machine.

From this point of view we divide all the SAKO sentences into the
following categories:

<u>DECLARATIONS</u> - they give information on the structure of
the program and determine the meaning of the symbols used.Declara-
tions perform their role while introducing the program into the ma-
chine,but they are omitted while the program is being performed by
the machine.

<u>INSTRUCTIONS</u> - determine the work of the machine while per-
forming the program.

<u>COMMENTARIES</u> - have only an explanatory character for the
programmer,and they are omitted while introducing as well as perfor-
ming the program by the machine.
The list of SAKO declarations and instructions is given on a separa-
te table.
One ought to differenciate the sequence of writing instructions by
a programmer from the sequence of performing them by the machine.
They are in general different.
The sequence of writing instructions corresponds to the sequence of
introducing them into the machine.

The sequence of performing instructions is determined as follows:

- the first instruction performed by the machine is always the first instruction written in the program,

- after having performed any one of the instructions which is not the control one, the machine takes the next instruction, i.e. the mearest in the written sequence,

- the transfer to an instruction which is different in sequence may take place, only after a control instruction and conditions being fulfilled, and in a way determined by such an instruction.

The last sentence written in a program is always the declaration END, which signalizes the end of introducing the program into the machine. The instruction STOP is last to be performed, and it signalizes the end of the executing of the program and the stoping of the machine.

Bigger programs written in SAKO must be divided in CHAPTERS which, together with the corresponding data can be included simultaneously in the internal machine storage.

The SINTAX of SAKO sentences is the following:

The sentences SAKO may be built out of following characters:

1. Latin alphabet containing 25 letters A,....,Z
2. Digits 0,....,9
3. Characters of operations and relations + - · ⁄ × = ≡ >
4. Separating characters ., : ( )
5. Space / space between characters/

The above given characters are available in Creeds Teleprinter; moreover, characters and were obtained by a change of types and .

We call some character sets having a determined structure / with the exactness up to space/ e x p r e s s i o n s, and we ascribe them individual meanings. For example:

- numbers,
- variables and functions,
- arithmetical and Boolean expressions.

We call some determined character sets / with the exactness up to space/ p h r a s e s of the SAKO language. For example:

READ
NEXT
~~WHEN~~ IF

Some character sets built frecly but with a synomymously determined beginning and end we call t e x t s. For example:

- commentaries
- texts for writing on the output together with computing results.

## LIST OF SENTENCES SAKO

1. Declarations

   1. CHAPTER: name
   2. BLOCK: list of blocks
   3. STRUCTURE: list of blocks
   4. TABLE: name
   5. H INTEGER ~~INTEGRAL~~: list of variables and blocks
   6. PARAMETER SCALE $n$ .
   7. LANGUAGE SAS
   8. LANGUAGE SAKO
   9. END

2. Control Instructions

   1. JUMP TO $\alpha$
   2. JUMP ACCORDING TO I: list of labels
   3. GO TO CHAPTER: name
   4. IF A=B: $\alpha$ ,OTHERWISE $\beta$
   5. IF A B : $\alpha$ ,OTHERWISE $\beta$
   6. IF OVERFLOW : $\alpha$ ,OTHERWISE $\beta$
   7. IF KEY $n$ : $\alpha$ ,OTHERWISE $\beta$
   8. REPEAT FROM $\alpha$ : $A = B(C)D$
   9. STOP

3. Drum Instructions

   1. READ DRUM FROM I : list of variables and blocks
   2. WRITE DRUM FROM I : list of variables and blocks

4. Input-Output Instructions

   1. READ : list of variables and blocks
   2. READ LINE : name of block
   3. PRINT $(n.m)$: list of variables
   4. PRINT LINE : name of block
   5. PRINT WORD : list of variables
   6. TEXT :
   7. TEXT LINES $n$:
   8. SPACE $n$
   9. LINE $n$

5. Arithmetical and Boolean Instructions

   1. $A = B$
   2. $A \equiv B$
   3. SET SCALE $N$
   4. H POSITIONS CHANGE SCALE ~~FROM n TO~~ $m$ : list of variables and blocks

## 6. Subroutines

1. SUBROUTINE : (list of results) = name (list of arguments)
2. (list of results) = name (list of arguments)
3. SUBSTITUTE : name (list of arguments)
4. RETURN

## Indications

| | |
|---|---|
| $n, m$ | natural numbers |
| $A, B$ | variables or expressions |
| $N, M$ | natural variables or numbers |
| $\alpha, \beta$ | labels |

The SAKO sentences are built in a form of [illegible] expressions and texts. For each [illegible] there are exactly fixed [illegible] types of [illegible] tions and sequence in which the elements [illegible]

Each SAKO sentence begins with a new line. The [illegible] are occupied by a sentence, results exclusively [illegible] which determines synonymously the end of the [illegible] of SAKO sentences is enclosed in one line.

Although the rules of building types of SAKO sentences [illegible] at heterogenity of their structure, the majority is built according to the following system:

phrase - operational part of instruction or declaration,
expression - parameters of sentences,
phrase - in the form of colon,
expression - arguments of instruction or declaration in the form of a list of variables.

This is an example of the instruction built accordingly to the scheme:

```
PRINT (6.4): A, B, C
```

The ressemblance of the building of SAKO sentences enables their being remembered by the programmer, also the differentiation by the machine.

## DECIMAL REPRESENTATION OF NUMBERS IN SAKO

### 1. Introductory remarks

XYZ and ZAM II are binary, fixed-point machines which, however, have the possibility of choosing the computation scale by means of shifting the double length accumulator after multiplying or before dividing . Based on these features the way of representing numbers and fulfilling arithmetical operations in SAKO language was accepted.

It is necessary to remark that it is generally most convenient for programming to accept the floating point arithmetic, and its simulation in fixed-point machines is possible by means of suitable subroutines. However, as it leads to a serious reducing of the speed of the machine, this method would be contrary to the first point of the assumptions given in the introduction, and therefore, it was not accepted in SAKO.

### 2. Fraction numbers.

Though the machine XYZ and ZAM II work in a binary system for the majority of computing a decimal interpretation of all the numbers which arise in a program or which are treated by the program may be assumed. Then we accept, that one storage cell is needed to write a fraction number in the machine, and we imagine that this cell is composed of 12 positions.

In the first of them is written the number character, in the next positions are placed sequently decimal digits and the point dividing the integral part and the fraction number part, for instance:

```
+  3  7  5  1  .  6  4
   0  2     3
```

The position number in which the point is written is named <u>d e c i m a l  s c a l e</u> of the fraction number. For instance, in the above given example, the given number is written in scale 4.

The following are the basic rules for the operating with fraction numbers:

1. Arithmetical operations on fraction numbers may be executed only when both operation components are written in the machine storage in the same scale.

2. The scale of the result of the operation on fraction numbers is always the same as the scale of both operation components.

3. The scale in which the arithmetical operations are performed is determined by the instruction SET SCALE, and it must be in accordance with the scale of writing all the fraction operation components. In case these scales disagree the result we obtain is wrong.

4. The scale of fraction numbers written in the program is stated by the last preceeding declaration PARAMETER SCALE.

5. If, resulting from the operation, we obtain more significant digits before the point than it is permissible in the accepted scale, an overflow arises accompanied by a wrong registration of the number in the storage cell.

As it results from the above given presenting of the machine storage cell, we can always write, in the scale $N$, the number having till $N$ significant decimal digits before the point. However, there really exists a possibility of some extension of these limits as it will further results from this article.

6. The arising of an overflow is signalized in the machine by digit automatically written into a special register called <u>o v e r f l o w  i n d i c a t o r</u>. Its state, which may be number 0 or 1, may be discovered by the instruction IF OVERFLOW which writes at the same time the number 0 into the overflow indicator.

The number scale written in the storage may be changed by the instruction CHANGE SCALE.

3. <u>Integer Numbers</u>

For writing an integer number in the machine, we use half of the machine storage cell. We imagine that this half consists of 6 positions. In the first of them is written the number ~~character~~, in the rest - digits of the integer number. The less significant digit of the number is located in the last position.

sign of a

The setting of the scale does not influence the registering of the integer number.
Operating with integer numbers in the SAKO language is based on the following rules:

1. The result of adding,substracting,multiplying and raisig to a power of two integer numbers is always integer.

2. While dividing integer numbers or,if at least one operation component is a fraction number while adding,substracting or raising to a power,the result is a fraction number written in the scale determined by the last instruction SET SCALE.

3. When the integer number overflows the admitted interval, there arises an OVERFLOW like in the case of fraction numbers.

4. Example.

The use of the instruction SET SCALE and of the declaration PARAMETER SCALE,besides other SAKO sentences,may be illustrated as follows:

Let us assume that 101 product values ought to be computed

$$y = a_0 + a_1 \cdot X + a_2 \cdot X^2 + a_3 \, X^3$$

for the given values $a_K$ of absolute values smaller than 1 and for varying from 0 every 0.01 to 1 .

The suitable program in SAKO may be written as follows:

```
        K)COMPUTING OF POLINOMIAL VALUES
        PARAMETER SCALE 1
        TABLE (4):A
        0 38481   -2.48521
        -0.05846   0.99846
        INTEGER : K
        SET SCALE 1
     *2) Y=0
     *1) Y=Y*X+A(K)
        REPEAT FROM 2 : K=3(-1)0
        LINE
        PRINT (6.2):X
        PRINT (6.4):Y
        REPEAT FROM 1 : X=0.0(0.01)1.0
        STOP
        END
```

The meaning of separate sentences of this program is the following:

The first sentence is the commentary which has no significance for the run of the program.

The second sentence is a declaration which establishes the scale of reading in into machine fraction numbers written further in the program.

Declaration TABLE determines the four numbers written under it as components of vector A .

All the given numbers will be stored in the machine memory in scale 1 in accordance with the lately written declaration PARAMETER SCALE.

The declaration INTEGER determines K as variable,accepting integer values.

The instruction SET SCALE 11 establishes the scale in which the following computings will run.

The next instruction giving Y the starting value equal to 1.0 is labelled number 2 .

The asterisk on the left of the label means the beginning of the cycle REPEAT.Generally before each label we must put exactly as many asterisks as instructions REPEAT refer to this label.Each label must begin with a digit.
The two next instructions cause the computing of the polynomial values according to Horner's scheme.The computing will be done in scale 1 ,in conformity with the lately performed instruction SET SCALE.

The three next instructions cause the printing in one line of values X and Y in a sequence.For printing X there are reserved 6 positions before the decimal point,and 2 after the point.This gives,together with the point,9 positions.For printing Y there are reserved 11 positions,moreover the results are rounded to four digits after the point.Instruction LINE causes the transfer to the next line.

The following instruction REPEAT induces the performing of the program interval from label 2 to this instruction in sequence for values X varying from 00 every 0.1 to 10 .

Instruction STOP causes the stopping of the machine.
Declaration END signalizes the end of the program.

## 5. <u>The binary writing of numbers.Boolean words.</u>

### 1. Binary word.

The decimal interpretation of the storage places was given above. However,the binary interpretation and the use of it is nearer to the real machine construction,and may give additional advantages in many cases.
In binary interpretation each storage cell is treated as a set of 39 positions containing one of two digits 0 or 1 .Such a set we shall call b i n a r y w o r d or shorter - w o r d.

Depending upon interpretation we can treat the word as a fraction number,integer number of Boolean word.

### 2. Fraction numbers.

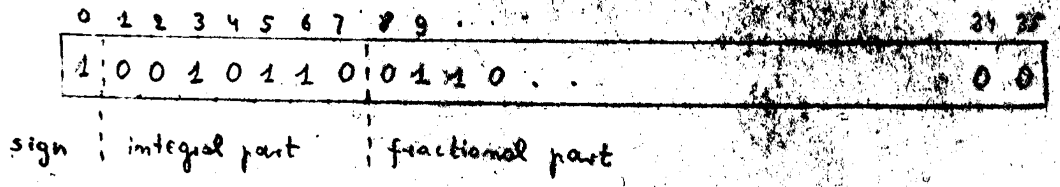
We write fraction and integer numbers in a binary word using the binary system to represent them.

Digit in zero position of the word determines the sign of the number:

0 - plus number
1 - minus number

In the word the integer and fraction part of the number takes two parts immediately following each other.The position number in which the last digit of the integer part is placed is called b i n a r y s c a l e .

The following example



represents the number -22.375 ,written in binary scale 7 .

If the fraction part begins with position 1 ,we say that the number is written in scale 0 .If the integer part finishes in position N , we say that the number is written in binary scale N .

The binary scale is not marked in the word while writing fraction numbers,but it is taken into account in the object program by a suitable way of shifting the double length accumulator.These shiftings are caused by the machine instruction EXECUTE INSTRUCTION LOCATED IN THE CELL n directly after every instruction to multiply,or EXECUTE INSTRUCTION LOCATED IN THE CELL m placed directly before every instruction to divide.The cell with the address n contains the instruction LONG SHIFT LEFT p ,and the cell with the address LONG SHIFT RIGHT p .By changing the number p in storage cells with addresses n and m ,we change the scale of all following computings.

In the SAKO language the binary scale may be signalised to the machine by instruction SET SCALE or by declaration PARAMETER SCALE, for instance:

```
SET SCALE 12B
PARAMETER SCALE 8B
```

The letter B written after the number determining the scale signalizes to the machine that this instruction or declaration concerns the binary scale.
To each decimal scale determined previously,corresponds a binary scale according to the following table:

| Dec. Scale | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Bin Scale | 0 | 4 | 7 | 10 | 14 | 17 | 20 | 24 | 27 | 30 | 35 |

So the numbers written in decimal scale 2 :

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| - | 2 | 2 | . | 3 | 7 | 5 | 0 | 0 | 0 | 0 | 0 |

in the machine will be represented in scale 7B in the way given in the previous example.

## 3. Integer number.

To write an integer number in the machine we use only half of the word i.e. 18 binary positions.Therefore,in one machine storage cell we can simultaneously write two integer numbers.

The first position of the half word serves for writing the integer number sign. The last significant binary digit is written in the last of the 18 positions.
While performing operations on integer numbers, the machine treates them as if they were written in 36 positions, where half of the word containing the integer number was located in positions from 0 to 17, and only zeros in positions 18 to 35.

## 4. Boolean word.

By Boolean word we understand a word whose digits 0 and 1 are treated as elements of the two-element Boolean algebra. Especially every fraction and integer number written in the machine can be treated as a Boolean word.
We write Boolean words in octal notation by expressing each sequence of three binary digits as digits from 0 to 7.
For instance:

$$776.000.000.377$$

represents a Boolean word in which 8 first and 8 last bits are one, and the remaining bits are zeros.
In accordance with operations in Boolean algebra, we introduce for Boolean words the ideas of identity and operations of the sum, product and negation. Moreover, we determine the idea of c y c l e s h i f t i n g of a word. The sense of these operations is the following:

Two Boolean words $A$ and $B$ are identical:

$$A \equiv B$$

if and only if all digits of one word are identical with the corresponding digits of the second word.
The result of the s u m of the Boolean words $B$ and $C$:

$$A \equiv B + C$$

is the Boolean word $A$, in which on every position is the digit 1 if and only if the digit 1 is on this position at least in one word $B$ or $C$.
We likewise determine the Boolean product and negation

$$A \equiv B \cdot C \quad \text{and} \quad A \equiv - B$$

The result of a cyclical shift of one position to the right of the Boolean word $B$

$$A \equiv B * 1$$

is the Boolean word $A$, which arises in such a way, that every digit being in the word $A$ on position $\alpha$ ($\alpha < 35$) is in the word $A$ on position $\alpha + 1$. The digit being in word $B$ on position 35 transfers in word $A$ to zero position.
The result of a cyclic shift of the word $B$ of $N$ positions:

$$A \equiv B * N$$

is the word $A$, arising from the word $B$ by $N$ times cyclic shift of each ~~one~~ position of this word to the right.

We likewise determine a cyclic shift of the word $B$ by $N$ positions to the left:

$$A \equiv B * (-N)$$

We can obtain a complicated Boolean expression by composing Boolean operations like arithmetical ones.
Out of two operations in Boolean expression one performs at first this, which is enclosed in a bigger amount of parenthesises. In case of an equal amount of parenthesises the operation must be executed in the following hierarchy:

1. Shifts,
2. Products,
3. Sums and negations

The operations enclosed in the same amount of parenthesises and of the same hierarchy ought to be executed sequently from the left to the right side of the Boolean expression.
To illustrate the above mentioned rules let us take the following example. We assume to make a Boolean product of the positions $0-9$ of the word $B$, and positions $10-19$ of the word $C$, and add to it the negation of positions $20-29$ of the word $D$. The result is to be placed in positions $26-35$ of the word $A$, the remaining positions being zeroed.
The above given operation may be expressed by the formula:

$$A \equiv (B * 20 \cdot C * 10 + (-D)) * 6 \cdot 000.000.001.777$$

One ought to emphasize once more, that since numbers in the machine are written by use of 36 digits 0 or 1, each number may be treated as a Boolean word, and on the contrary. The value of the variable $A$ in this example is the 36 bits set, which may be trated equally as a Boolean word, or as a number.

Discrimination of Boolean and arithmetical operations in the formula follows from the characters $\equiv$ or $=$.

## BLOCK STRUCTURE.

The declaration BLOCK, containing always constant parameters, determines the structure of all blocks mentioned in the list, and at the same time, it reserves a suitable amount of storage cells for these blocks.
The declaration STRUCTURE, in which variable parameters may arise, induces the change of the block structure introduced previously into the declaration BLOCK, but it does not reserve any storage cells. A new block structure must allow its full location in the previously reserved storage cells.

In instructions using the block structure always oblige a structure determined in declaration BLOCK or STRUCTURE written lately before the instruction.

Example:
Let us examine the following program interval:

```
BLOCK (100) : A
INTEGER ` N, M
READ : N, M
STRUCTURE (N,M) A
READ : * A
........................
```

The first declaration BLOCK determines the block as a vector,at the same time it also reserves 100storage cells for its component location.
The declaration INTEGER and the instruction READ cause the read in of integer numbers indicated as N and M into the machine.

The declaration STRUCTURE determines the new structure of the block A and it defines it as a matrix of size NM .This product must be smaller than 1oo,because in an opposite case it would not be possible to locate this matrix in the storage places reserved previously.

The next instruction READ induces the read in of all elements of the matrix A into the storage etc.

The asterisk before A in instruction READ shows that A is the block symbol.

SOME INPUT AND OUTPUT INSTRUCTIONS.

1. READ.

The input data introduced by this instruction into the machine must fulfill following conditions:

1. Integer numbers are written as a set of no more than five digits.

2. Fraction numbers are written as a set of no more than ten digits separated by decimal point.

H Negative
H Positive

3. ~~Minus~~ numbers are proceded by the character - .~~Plus~~ numbers can,but they musst not be preceded by the character + .

4. Boolean words are written in octal form and preceded by the character ≡ .

5. The space cannot appear between characters making the registration of the same number or Boolean word.

6. Several numbers or Boolean words may be written in one line,but at least one space must follow every number or word.

7. Separate numbers or lines may be provided with comments which are not read in into the machine storage.Every character set beginning with a letter,and finishing with the character ' or ⊢ will be trated as comment.The comment may occupy the whole line.

8. Every block written on an input data form must begin with a new line.Also after finishing the block,the writing of further data ought to begin with a new line.

Let us assume that we have as input data of the properties investigation program of rectangular matrices:

- Integer number showing the matrix range,
- Block of fraction numbers being matrix elements,
- Boolean word, helpful in some logical operations.

This data can be written in the following form:

```
MATRIX RANGE  = 2
MATRIX ELEMENTS A:
 0.43    7.21
-0.28    3.88
MASK : =774.000.000.000
```

The above given data may be read by the instruction

```
READ : N, *A, M
```

which will induce the reading of number, matrix and Boolean word in sequence.

2. Read line.

This instruction causes the read in into the machine of successive characters making input data, beginning from the extremely left position of the nearest line. Every successive line character, expressed by six bits, is written in the last six positions of the successive word of the block named in this instruction. The remaining positions of the word will be zeroed. The reading of the line finishes at the line end or if there are no more words in the block.

3. Print line.

Induces the writing of the line beginning from the extremely left position of the nearest line. Every successive line character is determined by the content of six less significant positions of the successing words of the block given. The end of printing is on the end of the block or at the character CARRIAGE RETURN.

The above mentioned two instructions allow to accept the input data or to give results by a freely accepted code. One of the applications may be the texts reading by the machine, their analysis and processing with the use of Boolean formulas, and then editing the processed texts.

REMARKS ON TRANSLATION.

The translator consiceness and its short time of work are the additional aspects not mentioned above, but they were taken into account while designing SAKO. At the beginning we were afraid that for such a simple machine as XYZ, for which was SAKO designed at first, the elaboration of the translator might be very difficult. But it appeared that the simplicity of XYZ involved also the simplicity of the translator. This became quite clear in the machine ZAM II, which design was based on the knowledge of SAKO, and which took into account the needs of its translation. The translator SAKO, including subroutines of the language and dictionnaries, contains for this machine less than 3.000 machine words /one word contains two one-address instructions/. Object programs given by this translator are almost optimal.

The translator SAKO was made with the use of the SAKO language. Separate parts of this program were written in SAKO and manually translated into SAS language. This translation was done quite mechanical on the basis of SAKO-SAS transfer rules. The obtained program was additionally optimalized. This method considerably shortened the time spent for writing the translator.

The capacity of SAKO language for testing such problems as data processing was simultaneously confirmed.

## CONCLUSIONS

Taking into account specifical features of the machine while designing automatic programming system, one may cover almost the full set of numerical and logical problems, for which this system provides the efficient use of the machine. Especially including of Boolean formulas, based on machine words construction, enables the programming of every type logical operations. In the concrete case the system SAKO eliminates almost the need of programming in the machine language for XYZ and ZAM II, for which it was designed.

## ACKNOWLEDGEMENT

The elaboration of SAKO was possible thanks to the efficient work of the whole staff of Automatic Coding Group of the Department of Programming of the Institute of Mathematical Machines of the Polish Academy of Sciences. Special investment while designing the system was given by A.MAZURKIEWICZ, J.SWIANIEWICZ, J.BOROWIEC, P.SZORC, and many others.

INSTITUTE OF MATHEMATICAL MACHINES - WARSAW 1959.

REFERENCES

1. Werner Buchholz: The system design of the IBM 701 computer, Proc. of the IRE, October 1953, s. 1262-1275.

2. J.W. Backus, R.J. Beeber and others: The FORTRAN Automatic Coding System, Proc. of the Western Joint Computer Conference, 1957.

3. R.A. Brookner, B. Richards, E. Berg, R.H. Kerr: The Manchester Mercury Autocode System, Computing Machine Laboratory, The University, Manchester, May 1959.

4. A.J. Perlis, K. Samelson: Report on the Algorithmic Language ALGOL, Numerische Mathematik, Bd. 1, s. 41-60, 1959

5. A. Mazurkiewicz: Arithmetical Operations and Subroutines in SAKO, ZAM, Warszawa, 1960.