PROCUREMENT EXECUTIVE, MINISTRY OF DEFENCE

ATOMIC WEAPONS RESEARCH ESTABLISHMENT
Building E3
Aldermaston, Reading, RG7 4PR
Telephone Tadley 4111 (STD 073 56 4111)
Telex 848104/5

Ext: 7818
Our Ref: SSCM 92/7/76
Your Ref:
Date: 29th July 1976

Professor Donald E. Knuth
Computer Science Department
Stanford University,
Stanford, California 94305

Dear Don,

I don't mind at all your requests for historical material: in some cases I can satisfy you.

I am sending you by surface mail the code listing of the S2 compiler and two copies of the Users' Manual which describes what the compiler could do. This compiler was probably the first to exploit a large immediate access store on a computer, in order to get high compilation speed. As you will see from the acknowledgements in the Users' Manual it was a team effort in which the now fashionable "Head Programmer Team" system was used. Many of the implementation features are described in Hopgood's book "Compiling Techniques" published by MacDonald-Elvisier. Hopgood was one of the constructors of the compiler.

As you are interested in historical material I am also sending you the Users' Manual for a compiler contemporaneous with Fortran I which I distributed through SHARE in 1958. It is possible that some of the concepts in that system have historical significance, for example, it contains an elementary form of the "picture" concept of COBOL. It is interesting to recall that in 1958 one could distribute a compiler to a users' cooperative. This compiler was used by those SHARE members who had too few magnetic tapes to run IBM's Fortran system. I believe it persisted in use in some American IBM 704 installations well into the '60s under some other name, having been taken over by a group of three or four installations. This is really only rumour to me so I cannot give you any supporting evidence.

Yours sincerely,

Alick

Alick E. Glennie

```
      ABC FUNCTION SQRT
      ABC OPERATIONS INPUT OUTPUT
      ABC DIMENSION A(11)
      ABC SECTION, LOOP
      ABC   X = A(I+1)
      ABC   Y = SQRT(MOD X) + 5 X X X        also SQRT X + 5 PWR(X,3) with mo
      ABC   IF (Y-400) IS POSITIVE GO TO BIG
      ABC   OUTPUT (PRINT, F1, I, X)
      ABC   GO TO LPDONE
BIG   ABC   OUTPUT (PRINT, F2, I)
LPDONE ABC  END OF SECTION LOOP
F1    ABC   FORMAT
            I =.□I, A(I) = .WW.WWWWWW F
            blank card
F2    ABC   FORMAT
            I =. I, A(I) = TOO LARGE
            blank card
START ABC   IN (CARDS, 11, A)
      ABC   DO LOOP I = 10, -1, 0
      ABC   STOP
      ABC   COMPILE
      END   START
```

comments would be done in SAP form, but the ABC manual has no examples!
so comments probably were not used in a program like this

1958 falls outside my time frame, so I'm not including this in the article —
I wrote it just for my own "amusement".

13 March 2003

# The ABC automatic coding system - AB C.

Miss J. Elliott and A. E. Glennie.

June 2nd., 1958.

Atomic Weapons Research Establishment,
Aldermaston, Berkshire, England.

This system generalises the SAP symbolic code for writing programmes for the IBM 704. The system will accept programmes written in a mathematical notation or in the normal SAP code, or in a mixture of the two.

The system includes an extensive library of pre-written subroutines for the evaluation of functions and for many other programme operations.

The system itself requires a 704 with a minimum of 8192 core storages and 3 tape units, but will write programmes for any 704. The UA SAP assembly programme is used as the last phase in the production of a programme.

## Table of Contents

Table of Contents (Cont.)

## 1.0 Introduction.

The notation normally used in coding for the IBM 704 (the SAP symbolic code) involves writing down each elementary order of the programme, i.e. each machine order. This makes the coding notation completely general, in the sense that it can be used to write any possible programme.

But programmes usually have some basis in mathematics or data processing, and it is convenient if the code includes the commonly used mathematical operations. The ABC automatic coding system augments the normal language with mathematical notation for formulae evaluation and certain functions. English descriptive phrases specify organisation.

The principal routine of the ABC system is a translator whose function is to translate from the special ABC notation into the SAP symbolic notation. Any SAP language presented to the translator is transmitted unchanged.

The translation process includes the generation from the ABC library of any subroutines which have been requested. When the translation is completed, an assembly process begins, using UA SAP 1 and 2. ONLY

The basic form of input to the ABC programme is the normal SAP card form. Location symbols may be punched in columns 1-6. Any of the normal SAP operations may be punched in columns 8-10, in which case columns 12-72 will be treated in the normal SAP manner. If the pseudo-operation ABC is punched in columns 8-10 the contents of columns 12-72 are treated as a statement requiring translation by the ABC translator. An ABC statement may not exceed the capacity of one card.

## 2.0 General description of the ABC system.

The ABC system consists of four parts, which exist on tape 1 in the following order:-

(1) A card to tape conversion programme.
This transcribes a programme from cards to tape 4. If tape 4 has been prepared on off-line equipment, this section must be by-passed.

(2) The ABC translator.
This accepts input data from tape 4 in SAP or ABC form. It transcribes the SAP data to tape 3 without change and translates the ABC notation to SAP symbolic notation, which is then written on tape 3.
If any subroutines are required from the system library, they will be punched out by the ABC translator to absolute positions from the highest address of core storage downwards. An upper one-card loader precedes these subroutines.

(3) The Library section.
This consists of a large number of useful subroutines.

(4) The Assembly programmes.
When the ABC translator has completed its action, tape 3 will contain a SAP symbolic programme, part of which may be original SAP coding and part a translation to SAP symbolic of the ABC notation. The assembly process then provides the absolute binary programme, apart from the binary subroutines already provided by the translator from the library section.

The whole process of translation and assembly is a three-pass process. The first pass is the ABC to SAP translation and the other two passes are those of the UA SAP assembly.

A listing of the programme may be obtained in the usual manner. It will not include the routines obtained from the library section. The original ABC statements will be printed as remarks by the assembly programme, the SAP instructions produced by each ABC card being listed in the normal manner. A symbol table may also be produced if required.

## 3.0 THE ABC NOTATION.

### 3.1 Symbols and their definition.

A symbol is a combination of any number of alphabetic or numeric characters (not special characters) of which the first, and at least one of the last 6 if more than 6 are used, must be alphabetic. If more than 6 characters are used, only the last 6 will be regarded as a symbol.

In the ABC notation symbols have several uses; they may be the names of variables, functions, arrays or operations. In order that the translator may be able to recognise the different uses of symbols, certain definitions must be made about them. These definitions are made by DEFINITION cards.

A symbol for which no definition is made is the name of a single variable in the floating point mode.

(1) Fixed point variables.
Fixed point variables take integer values of magnitude less than 2097152 and appear in the registers of the machine with the binary point at the extreme right.
To define a single variable or an array to be in the fixed point mode, write

ABC INTEGER I

This defines I as a fixed point variable, or a set of fixed point variables if I is the name of an array.
It is possible to define several variables on the same card.
e.g.

ABC INTEGERS I J K IZ

which defines I, J, K and IZ as fixed point variables.
(the symbols in the definition may be separated by blank characters or single commas.)

(2) Functions.
In the ABC notation two sorts of functional operation occur; these are called FUNCTIONS and OPERATIONS.
Functions are defined as functions of a single algebraic variable.
i.e. with the form F(x).
Operations are functions of more than one variable, which may be of more general type.
To define a symbol as a function name, write

ABC FUNCTION SIN

As with the definition of integers it is possible to define many symbols as function names on the same card.

(3) Operations.
A symbol is defined as an operation name by the notation

ABC OPERATION OP3

which defines OP3 as an operation.

(4) Externals.
We now introduce the concept of externally defined symbols. These are symbols whose ultimate absolute location in storage is to be allocated by the programmer and not by the ABC translator.
A variable not defined as an external will be allocated storage space by the translator.
Similarly a function or operation not defined as an external is assumed to be a subroutine of the system library and the translator will allocate storage for it and issue an absolute binary programme for it (if it exists on the library tape).
A variable defined as an external must be allocated storage by SAP coding; a function or operation which is defined as an external must have programme provided by SAP coding or by other means.
The notation for definitions is

ABC EXTERNAL A B C

As with the definitions for integers, functions and operations several symbols may be defined on the same card.

(5) Arrays.
The ABC notation allows the use of one-dimensional arrays. In order that storage may be allocated to an array it is necessary to specify the number of elements in the array. This is done by using the definition card

ABC DIMENSION A 10

In this example the array A is defined to be of 10 elements, which are arrayed in storage in locations A-1, A-2,..., A-10. Thus the array is stored backwards in storage, the most suitable arrangement for use with the 704 index registers. Every symbol specifying an array requires a dimension statement.
Note that in contrast to the other definitions, only one definition of dimension may be made per card.

The definition statements must appear in the programme before symbols to which they refer are used. Indeed it will be convenient to collect them at the beginning of the programme, where they form a glossary of the symbols used.

The same symbol may appear in several definition statements; for example if it is to be defined as an externally defined array of fixed point numbers, it will appear in three definition statements.

Example of a glossary of definition statements:-

```
ABC INTEGERS, A, B, C, D
ABC FUNCTIONS SIN, COS
ABC OPERATIONS INPUT, OUTPUT
ABC EXTERNAL X
ABC DIMENSION X, 25
```

these define A, B, C, D as integers; SIN and COS as functions and so on.

## 3.2 Arrays and Subscripts.

The ABC notation allows the use of one-dimensional arrays. As explained previously, any name used for an array must be defined in a DIMENSION statement.

An element of an array may be used as a variable in arithmetical formulae if it is written in the form X(I), where X is an array name and I is a subscript expression. This is the only legal way in which an array name can appear in arithmetical formulae; the subscript is obligatory.

There are two methods of subscripting, the ABC and SAP methods. The former is for normal use when the system is used as an automatic coder, the latter is for use when SAP coding is used to control the counting and ABC notation for writing the arithmetical formulae.

The ABC subscript forms are X(I), X(C), X(I+C), X(I-C) where I is a fixed point variable and C is a constant integer.

The SAP subscript form is X(A,B) where A is absolute or symbolic and may be preceded by + or -, and B is absolute and may be one of 1, 2 or 4. The comma is obligatory.

The table below shows the translations into SAP code.

| Subscripted variable | A,T,D field of translated order |
|---|---|
| X(C) | X-C |
| X(I) | *X,IR |
| X(I+C) | *X-C,IR |
| X(I-C) | *X+C,IR |
| X(A,B) | X+A,B |

* When the evaluation of a formula with array elements is done, an index register is filled with the variable part of the subscript by the order LXA I,IR where IR is 1 or 2. The compiled programme will use the current value of the subscript in all circumstances.

Examples of formulae with subscripted terms:-

```
ABC DIMENSION X 25
ABC DIMENSION Y 25
ABC INTEGERS I,J,K
ABC X(I)=X(J)+Y(K)
ABC Y(3)=3.542
ABC X(K+1)=Y(J-1)*Y(J+1)
```

The last example may be written as X(K+1)=Y(J-1)Y(J+1)

## 3.3 Mathematical formulae.

In this section we consider the formulae which may be written in ABC notation. At first, only formulae involving the operations of addition, subtraction, multiplication and division will be described.

The general form is

X=E

where X is a single variable or element of an array and E is an expression of variables, array elements or constants formed by using the elementary arithmetic operations. This formula means "replace the value of X by the value of E".

### 3.31 Variables.

These are denoted by symbols, and may be fixed or floating point according to the definitions made.

### 3.32 Constants.

Two forms of constant are possible, decimal constants and integers. Decimal constants are written with a decimal point (e.g. 1., 1.2) and integer constants are written without it.

Decimal constants are always stored in the floating point mode, but integer constants may be either mode, the choice being made by the translator according to context.

Decimal exponents such as are used by SAP (e.g. 1.2E-3) are not permitted by the ABC programme.

### 3.33 Expressions.

The expressions on the right hand sides of formulae are formed from variables, constants and the +, -, / and * signs. Brackets may also be used. The multiplication and addition signs may be omitted as in normal mathematics provided the symbols are distinct.

For example the formula A=+B may be written as A=B because the = sign separates the symbols A and B.

As examples of multiplication by implication consider A=3B (equivalent to A=3×B) and A=B C (equivalent to A=B×C). In the first case 3B is not a symbol, by definition, and it is recognised by the translator as a product. In the second case the factors B and C are separated by a blank and are therefore not the single symbol BC. Here multiplication is inferred.

The second case becomes more realistic when the second factor is an expression in brackets as in the examples:-

A=B(A+B) and A=(A+B)(A-B)

Provided all the symbols are separated by blanks, signs or brackets, all the usual omissions of + and × signs are permissible.

3.34 Order of operations.

In the evaluation of expressions the multiplications and divisions will be done before the additions and subtractions.

There is a convention about the use of the division operation. The division sign / applies only to the term to its right and the expression is evaluated as if /X were equivalent to multiplication by the reciprocal of X.

Any other meaning can be established by using brackets (which indeed should be used in all cases of possible ambiguity).

For example A+B×C/D×E-A1 is evaluated as A+(B×C×E/D)-A1

3.35 Fixed point arithmetic.

Any expression containing only fixed point variables and integer constants is a purely fixed point expression and will be evaluated entirely in fixed point arithmetic.

This has one result which should not be overlooked; in any division operation the quotient is an integer.

Consider the formula

X=A/B where A and B are fixed point, and X is a floating point variable.

The expression A/B is purely fixed point and is therefore evaluated as the integral part of A/B.

Where an expression contains repeated factors and divisors such as the expression A×B×C/D/E the numerator and denominator are formed before division. In this example the expression is evaluated as (A×B×C)/(D×E).

Where a fixed point expression contains no division operations, there is no ambiguity.

3.36 Floating point arithmetic.

Any expression which contains no fixed point variables is a purely floating point expression, and will be evaluated in floating point arithmetic.

3.37 Mixed arithmetic.

A mixed expression contains both floating point and fixed point variables, and is a permissible expression in ABC notation. The evaluation of a mixed expression is made partly with fixed point arithmetic and partly with floating point arithmetic. As with fixed point arithmetic, a difficulty of interpretation occurs with division involving fixed point numbers.

Any term of the form A×A×A/A/A, where A stands for a fixed point variable, integer constant or bracketed expression of these things, will be evaluated in fixed point arithmetic. If any of the factors are decimal constants or floating point numbers, then the term will be evaluated in floating point arithmetic.

The preceding paragraphs relate to the arithmetic used in evaluating the right hand side of the formulae. The result of the evaluation will be converted to fixed of floating point form according to the mode of the left hand side of the formulae (if it is not in the correct mode already).

Examples:-

In the following examples we use I, J, K and L for fixed point variables, X, Y and Z for floating point variables.

| | | |
|---|---|---|
| (1) | X=I | The fixed point variable I is converted to floating point and the result stored as X. |
| (2) | I=X | The integral part of X is converted to fixed point and stored as I. |
| (3) | X=2Y+3Z | This is a purely floating point formula, the constants 2 and 3 being converted to floating point constants by the translator. |
| (4) | X=Z/3 | As in the previous example, this is entirely floating point arithmetic. |
| (5) | X=Z/I | Mixed arithmetic; the divisor is first converted to floating point and the division made in floating point arithmetic. |
| (6) | X=I/J | In this example the right hand side is pure fixed point and the division is performed in fixed point arithmetic. Then the integral part of the quotient is converted to floating point and stored as X. |
| (7) | X=3I/J | or 3*I/J. This has a fixed point expression on the right and so X equals the integral part of 3I/J. |
| (8) | X=3.0I/J | or 3.0*I/J. This is a mixed expression evaluated in floating point arithmetic. X now takes the value 3I/J. |

3.38 Bracketed algebraic expressions.

In any formula, the expression on the right may contain algebraic brackets, as in the example

X=Z-Z1/(Z2-Z1) .

The contents of a bracket form an expression which may be pure fixed point, pure floating point, or mixed. The result of evaluating a bracket is a fixed point number only if the expression in the bracket is pure fixed point, otherwise the result is a floating point number. The mode of the result of evaluating a bracket will affect the mode of any expression containing the bracket, just as if it were a single variable.

3.39 The Modulus symbol MOD.

The symbol MOD is reserved for the function of taking the absolute value of a variable. Thus if X is a variable or the result of evaluating an algebraic bracket then MOD X means use the modulus (or absolute value) of X in the formula. The word MOD, which applies only to the variable to its right, must be separated from the variable. Example:-

*MOD X*Y means (MOD X)*Y; the other possible meaning is obtained by the expression MOD(X*Y).*

The symbol "MOD" does not affect the mode of arithmetic in any way.

## 3.4 Functions.

In the ABC notation, functions are defined as functions of one variable; all other functional operations fall into the class of "operations".

As already explained, all names of functions must be defined in a "FUNCTION" definition. Functions in the ABC notation operate in floating point arithmetic only and consequently their presence in an otherwise purely fixed point expression will convert it into a mixed expression.

The ABC notation for functions is the usual mathematical notation. For example, if COS is defined as a function name the permitted basic notations are COS(X) and COS X.

In the second notation some care is necessary. For instance, the symbols for the function and its argument must be separate. Also, if there are further terms to the right of X ambiguity may arise. This is best shown by examples.

| | | |
|---|---|---|
| COS X | is equivalent to | COS (X) |
| COS+X | | COS (X) |
| COS-X | | COS (-X) |
| COS X+Y | | COS (X)+Y |
| COS X*Y | | COS (X*Y) |

### 3.41 Rule for unbracketed arguments.

The argument of a function extends from the first variable after the function name to the first + or - sign encountered or to the end of the expression.

### 3.42 Iterated functions.

Functions may be iterated, as in the expression COS(COS(X)), and if the bracketed notation is used this is perfectly clear. If no brackets are used apply the above rule.

Thus COS X*COS COS X*Y/Z+W means COS(X*COS(COS(X*Y/Z)))+W

Normally examples as extreme as this will not be constructed, a more realistic example being LOG COS(X+Y). If there is doubt about the meaning of an expression it can always be clarified by the use of brackets or several simpler formulae.

3.43 Subscripting.

The arguments of functions may be subscripted in either the ABC or SAP form. However, index register 4 is used for entering all library subroutines. Therefore, if SAP coding has been used the contents of index register 4 are lost after the use of a function subroutine.

## 3.5 Operations.

In the ABC notation Operations are generalised functions. They appear in the form OP(A,B,C,...). The arguments must be separated from each other by commas and must be enclosed in brackets, as shown. Some operations may appear as components of mathematical formulae or English sentences, others must appear alone. The definition of each individual operation in the library list shows clearly to which class it belongs.

The arguments of operations may be of several types:-

(1) Constants.

(2) Algebraic variables which are used as input to the operation.

(3) Algebraic variables which are computed by the operation.

(4) Array names.

(5) Symbols denoting programme locations, such as error returns and subroutines for calculating auxiliary expressions.

(Detailed definitions of the operations used in the examples below will be found in Appendix 3).

Types 1 and 2 may be preceded by a + or - sign.
e.g. Z=POWER(X,-2.)

An arithmetical expression may be used instead of a single argument of types 1 or 2, provided the expression is enclosed in brackets.
e.g. I=MAX1((2J-1),-(3K/2))

Arguments of types 2 and 3 may be subscripted with SAP subscripts, provided that index register 4 is not used, since its contents are lost on entering the subroutine.
e.g. POLY1(6,X(-2,1),Y(I,2),Z)

Arguments of types 2 and 3 may use constant ABC subscripts without restriction.
e.g. MAX3(J,A,B(2),C(3),....)

Arguments of type 2 may be subscripted with variable ABC subscripts provided the argument and its subscript are enclosed by a pair of brackets.
e.g. MAX3(J,A,(B(I)),(B(J)),....,C)

There are two exceptions to this condition:-

i) If the last argument of an operation is subscripted, it and any other argument using the same subscript need not be enclosed in brackets.
e.g. MAX3(J,X,Y(I),(Z(J)),....,Z(I))

ii) Brackets are not necessary for any of the subscripted arguments of the operation OUTPUT.
e.g. OUTPUT(PRINT,F,A(I),B(J),....,C)

Arguments of type 3 (with the exception of subscripted arguments of the operation INPUT) may <u>not</u> use variable ABC subscripts.

Arguments of type 5 (excepting formats generated by the format generator) must be defined as external symbols. If this is not done duplicated symbols will appear in the final assembly.

Index registers 1 and 2 are preserved throughout an operation provided that any subscripts used are SAP subscripts. If ABC subscripts are used no guarantee can be given about the contents of index registers.

The programmer is responsible for ensuring the correct use of the arguments of an operation, for ensuring that algebraic variables or constants are in the correct mode, and that other symbols are of the correct type.

## 3.6 Translation directives.

### END

The SAP pseudo-operation END will occur in every ABC programme as the last statement of all. It states where the programme is to begin computing. It is also necessary to the ABC translator as an end signal, saying that the translation phase is finished, and the assembly phase (if required) may begin.

### COMPILE

This command is given at the end of the programme, immediately before the END card. It causes the translator to perform the following actions:-

(a) Place the constants and working space at the end of the programme, followed by

(b) The formats generated by the format generator,

(c) The variables and arrays for which the translator is to provide storage.

(d) Causes the 704 to punch a one card upper loader, followed by the subroutines for functions and operations.
The absolute locations for these subroutines are noted on tape 3, by the emission of SAP pseudo-operations in the form "NAME SYN Absolute location".

### WORKINGSPACE

This ABC pseudo-operation may be used to allocate a symbolic name to the block of constants etc. emitted by the translator. For this purpose use the statement

ABC WORKINGSPACE "Required name"

It is not normally necessary to use this facility, as a name is automatically provided by the translator - the symbol 0000( ) . The pseudo-operation WORKINGSPACE

is available so that the ABC system may be used to generate symbolic library programmes. It brings the working space symbol under the control of the user. If a WORKINGSPACE statement is used it should be placed at the beginning of the programme with the definition statements, before any arithmetical formulae.

## 3.7 Control statements in the form of English sentences.

The ABC programme contains a directory of English words which enables it to translate certain English sentences into 704 instructions. Only those words included in the directory (a list of which is given at the end of this section) are considered meaningful by the programme. Any other words will be ignored unless they are the names of symbolic locations, variables, etc. If the programme considers all the meaningful words in a sentence and finds that they do not translate into permissible machine instructions the entire sentence will be ignored.

There are several classes of sentence, the largest being control transfer sentences.

### 3.71 Unconditional transfer.

GO TO A

"A" must be a symbolic location defined somewhere in the programme or a decimal integer.

(The latter case is only likely to be useful when the location of a piece of programme is known to the programmer, e.g., the loader, or a piece of programme preceded by an ORG card.)

It is important to note that in all control transfer sentences the place to which control is to be transferred must be the last word in the sentence.

### 3.72 Conditional transfers.

IF IT IS POSITIVE GO TO A

"IT" always refers to the last variable computed. The word "IF" must be the first in the sentence.

If the variable to be tested is not the last one computed the sentence may be written:-

IF (B) IS POSITIVE GO TO A

The variable to be tested must be enclosed in brackets. It may consist of a single variable, as above,

an arithmetical expression of several variables or functions, or a single operation.

(A function is a library subroutine with only one argument, an operation is a library subroutine with more than one argument.)

e.g., IF (A-B SIN C) IS POSITIVE GO TO D
IF (MAX1(A,B)) IS POSITIVE GO TO D

Conditional transfers may also be made if the variable is negative or zero.

e.g., IF IT IS NEGATIVE GO TO A
IF (B) IS ZERO GO TO A

It should be remembered that in the 704 zero may be either positive or negative (depending on the order in which the operations from which it resulted were carried out) and therefore if zero, positive and negative values are all to be distinguished the test for zero must be performed first.

The flow of the programme may also be controlled by the status of certain triggers in the machine.

e.g., IF THE ACCUMULATOR OVERFLOW INDICATOR IS ON GO TO A
IF THE QUOTIENT OVERFLOWS GO TO B
IF THE DIVIDE CHECK INDICATOR IS OFF GO TO C

These three sentences turn off the indicator referred to when it is tested.

The sentence:-

IF THE MQ OVERFLOW INDICATOR IS OFF GO TO G

is not permitted.

It may be stated here that the terms

POSITIVE, PLUS, DOWN, ON

are considered synonymous, while the terms

NEGATIVE, MINUS, UP, OFF

are considered to have the opposite meaning.

Furthermore it will be noted that the "positive" words do not appear in the directory. The ABC programme works on the supposition that if one of the "negative" words does not appear in an "IF" sentence then the sentence has a "positive" meaning.

The word "NOT" may appear in an "IF" sentence in order to reverse the meaning.

e.g., NOT NEGATIVE means POSITIVE
NOT ON means OFF

Hyphenated words are not permitted. Therefore NOT ZERO represents "non-zero".

The set of 4 sense lights provide another means of controlling the flow of the programme. They may be turned on or off at will and their status may be tested. The testing instruction turns off the light tested. The lights may be turned off singly or as a set.

Examples of sentences involving sense lights are as follows:-

```
TURN SENSE LIGHT 1 ON
TURN SENSE LIGHT 2 OFF
TURN OFF THE SENSE LIGHTS
IF SENSE LIGHT 3 IS ON GO TO G
IF SENSE LIGHT 4 IS OFF GO TO H
```

It will be noted that the word "LIGHT" does not appear in the directory, but in sentences involving sense lights it must always be placed between the word "SENSE" and the number of the light.

The path of the programme may be controlled externally by the setting of the sense switches on the console.

e.g., IF SENSE SWITCH 5 IS DOWN GO TO D
IF SENSE SWITCH 6 IS UP GO TO E

Any English sentence may have a symbolic location.

e.g., START ABC TURN SENSE LIGHT 1 ON

A further type of conditional transfer is the "BRANCH" sentence.

e.g., BRANCH A,B,C,D

Control is transferred to B if A is 0 or 1
" " " " C " A " 2
" " " " D " A " 3
" " not transferred " A " greater than 3

The number of places to which control may be conditionally transferred is only restricted by the size of the card. Note that A must be an integer. If it is a subscripted variable it must be enclosed in brackets. No "meaningless" words are permitted in a "BRANCH" sentence, since all words other than BRANCH are treated as symbolic locations to which a transfer may be made.

3.73 Sections.

If portions of the programme require to be used in more than one place they may be written as normal closed subroutines. The ABC programme will produce a closed subroutine if the required computation is preceded by a card punched:-

SECTION XYZ

(Where XYZ is the name of the subroutine, i.e., the symbolic location of its first instruction. The name of the subroutine must be punched as the last word in the sentence, it may also be punched in the symbolic location field if desired.)

and followed by a card punched:-

END OF SECTION XYZ

(where XYZ must be the last word on the card).

In order to use such a subroutine a card punched

DO SECTION XYZ

must be placed at the required position in the programme.

After performing the operations specified in the subroutine control will be returned to the instruction following the "DO" sentence.

The "DO" sentence need not include the word "SECTION". It may not include any extra words, except in the case described below.

A SECTION may be performed a certain number of times before the programme proceeds to the next instruction.

e.g., DO SECTION XYZ J = A,B,C

means that
the 1st time XYZ will be performed with the variable J=A
" 2nd " " " " " " " " " J=A+B
" 3rd " " " " " " " " " J=A+2B
. .
. .
. .
" rth " " " " " " " " " J=A+(r-1)B=C

The programme then proceeds to the next instruction.

Care must be taken to see that the binary equivalent of A plus an integral multiple of the binary equivalent of B is exactly equal to the binary equivalent of C (no difficulty arises in the case of integers). The modes of the variables must be compatible. If J is an integer variable and any of A,B,C are constants they must be integers. If J is a floating point variable A,B,C may be written as integers. J may not be a subscripted variable, and A,B,C may only be subscripted if they are enclosed in brackets, e.g., (A(I)).

3.74 Imperatives.

PAUSE

will cause the machine to halt. Pressing the START button causes the programme to resume at the next instruction.

If this is written

PAUSE A

where A is a decimal integer, the binary equivalent of A will be in the address field of the storage register when the machine halts. No other words may be included in a "PAUSE" sentence.

The instruction

STOP

will cause the machine to halt in such a way that the programme cannot continue. It is intended as a final stop for a programme.

Most input-output operations are performed by a library subroutines but the following simple tape operations can be translated from English sentences:-

REWIND TAPE 6
BACKSPACE TAPE 4
WRITE END OF FILE ON TAPE 2

The numbers of the tapes may of course vary from 1-10. The tape number must be the last word in the sentence. It may not be a variable.

3.75 Formats.

A further word included in the directory is "FORMAT", which is used in connection with the decimal output operations "OUTPUT" and "OUT" in the library, in order to specify the layout of the printing, BCD tape writing, or punching.

Since FORMAT is closely related to the operations OUTPUT and OUT the description of its use has been placed in Appendix 3 together with the description of these operations.

3.76 Directory of English words.

| | | | |
|---|---|---|---|
| BACKSPACE | BRANCH | CHECK | DIVIDE |
| DO | END | FILE | FORMAT |
| GO | IF | LIGHTS | LITES |
| MINUS | MQ | NEGATIVE | NOT |
| OFF | OVERFLOW | OVERFLOWS | PAUSE |
| QUOTIENT | REWIND | SECTION | SENSE |
| STOP | SWITCH | UP | ZERO |

The words LIGHTS and LITES are synonymous, as also are OVERFLOW and OVERFLOWS, and MINUS, NEGATIVE, OFF, and UP.

Any words which do not appear in the directory may be placed in a sentence to improve its readability, provided they do not violate the restrictions placed on certain sentences.

In addition, any comments required may be punched in columns 12-72 of cards with the SAP pseudo-operation "REM". They will be listed but will not affect the programme in any way.

Sentences may be reduced until they contain only the necessary word from the directory.

Thus

IF OVERFLOW GO A

has the same meaning as

IF THE ACCUMULATOR OVERFLOWS GO TO A.

IF MQ OVERFLOW GO B

is equivalent to

IF THE QUOTIENT OVERFLOW INDICATOR IS ON GO TO B.

IF GO C

means IF THE ACCUMULATOR IS POSITIVE GO TO C

or IF IT IS POSITIVE GO TO C.

3.77 Glossary of English sentences.

This glossary is merely a guide to the type of English sentence which may be written. It is not intended to be a fully comprehensive list.

GO TO A
IF IT IS POSITIVE GO TO A
IF (B) IS NEGATIVE GO TO A
IF (A-B SIN C) IS ZERO GO TO D
IF THE AC OVERFLOW INDICATOR IS ON GO TO D

```
IF THE DIVIDE CHECK INDICATOR IS OFF GO TO E
TURN SENSE LIGHT 1 ON
TURN OFF THE SENSE LIGHTS
IF SENSE LIGHT 2 IS ON GO TO A
IF SENSE SWITCH 3 IS UP GO TO A
BRANCH A,B,C,D,...
SECTION XYZ
END OF SECTION XYZ
DO SECTION XYZ
DO XYZ J=A,B,C
PAUSE A
STOP
REWIND TAPE 6
BACKSPACE TAPE 4
WRITE END OF FILE ON TAPE 2
FORMAT
```

## 3.8 Sequencing in ABC programmes.

The programme starts at the formula named on the END card. If the END card does not name a formula and no SAP origin card precedes the first active formula, the programme will start at the first formula.

The formulae of the programme are evaluated in the order written, except when a control transfer instruction is encountered. The following list describes the sequencing of the various types of statements:-

(In normal sequencing, control proceeds to the next instruction)

| | | |
|---|---|---|
| (1) | Arithmetical formulae | Normal sequencing. |
| (2) | Unconditional transfer | Sequencing begins at the named formula. |
| (3) | Conditional transfer | If the condition is satisfied, sequencing begins at the named formula. Otherwise normal sequencing. |
| (4) | English imperatives | Normal sequencing, except of course STOP. |
| (5) | DO statements | Normal sequencing after action of the named section, and possible repeats. |
| (6) | SECTION | Normal sequencing. |
| (7) | END OF SECTION | Control returns to the formula from which the SECTION was entered. |
| (8) | Operations | Normal sequencing. Where one of the arguments is a formula symbol, control will be transferred to that formula. |

Definition statements and statements for the FORMAT generator are to be ignored in consideration of sequencing.

## 3.9 Constructing an ABC programme.

Storage used in ABC programmes:-

An origin card in SAP form may be used to specify the first location used in storing an ABC programme. If no origin is specified then storage begins at location 0. The storage used then runs upwards from this location, unless further origin cards in SAP form are used. This applies to the programmes translated from statements and to locations of variables for which the ABC translator allocates storage. Subroutines for functions and operations drawn from the ABC library occupy the upper end of the store, together with the one card loader.

The storage structure is then:-

```
0     Programme generated by the translator.
      Constants and working space allocated by the trans-
        lator.
      Format statements generated by the ABC translator.
      Single variables.
      Arrays.
      ............
      Subroutines from the ABC library.
-1    ABC upper one card loader.
```

This is the storage structure if no SAP origins are used. SAP origin cards may be used to allocate the programme proper to any location required, and to split it into sections. The section containing the constants, working space, formats, variables and arrays forms one continuous block which may not be divided. It is possible to control the allocation of storage for variables and arrays by defining them as EXTERNALS and arranging locations for them by SAP pseudo-operations. For single variables use a BSS pseudo-operation (e.g., X BSS 1); for arrays use a BES pseudo-operation with the appropriate dimension (e.g., A BES 8).

### 3.91 Example.

Example of a simple ABC programme. Details of the Input-Output operations and format generation will be found in Appendix 3.

To read a number X from a card and to print X and sin X.

```
          ABC FUNCTION SIN
          ABC OPERATIONS INPUT OUTPUT
          ABC INPUT (CARDS,X)
          ABC Z=SIN X
          ABC OUTPUT (PRINT,A,X,Z)
A         ABC FORMAT -
X=. .       F     Z=. .       F
          (blank card)
          ABC STOP
          ABC COMPILE
          END
```

Notes:-

(a) The first two cards define the functions and operations.

(b) A format is generated from the two cards following the ABC pseudo-operation "FORMAT". As written above, the printed output will contain some Hollerith characters as well as the values of X and sin X. A typical result might be X= 1.200000 Z= 0.9320390

(c) The card input data is X, which may be punched in any position on the card. It must be a floating point number, but the number of digits punched is at the disposal of the puncher, thus it could be punched as 1.2 in this example.

(d) The last two cards are the essential COMPILE and END operations.

(e) The storage used by the programme will be in two parts. In the lower positions in store, from location 0 onwards, will be the programme, the constants (if any) and working space, the format, the variables. At the upper end of storage will be the loader and the subroutines SIN, INPUT and OUTPUT. Certain dependent subroutines of INPUT and OUTPUT will also appear there.

## Appendix 1

### A1. Summary of the ABC notation.

(1) Definition statements.

```
INTEGER A
INTEGERS A B C

FUNCTION A
FUNCTIONS A B C

OPERATION A
OPERATIONS A B C

EXTERNAL A
EXTERNALS A B C

DIMENSION A,N
```

(The singular and plural forms of the first four definition statement types are completely identical in function. The translator does not insist on grammatical correctness.)

(2) Arithmetical Formulae.

The general form is L = E where L is a single variable or element of an array, and E is an expression of variables, array elements or constants. The expression may contain the operations +,-,× and / together with MOD and any named functions.

(3) Functions and Operations.

Functions are functions of one algebraic variable, and are computed in the floating point mode. Operations have more than one argument: these arguments may be constants, variables, array names or symbolic locations of parts of the programme. Operation statements are of the form L = OP(A,B,C,....) with the left hand side optional.

(4) Control statements.

These control the flow of the programme. They may be imperatives, or conditional transfers depending upon the various indicators which can be tested.

(5) Translation directives.

These are statements directing the translation programme only. They have no corresponding orders in the object programme.

| | |
|---|---|
| END | tells the ABC translator that the translation phase is finished, and tells the assembly programme to produce a transfer card. |
| COMPILE | generates the library functions, collects the working space block and allots storage. |
| WORKINGSPACE A | allots the symbol A to head the working space block. |

## Appendix 2

### A2. Input-Output Operations.

Among the many operations available in the ABC system perhaps the most important are those dealing with input and output, and with conversion between the decimal and binary scales.

#### A2.1 Input.

There are two operations for input, named INPUT and IN. Both take decimal data from cards (or tape written in the BCD mode), convert it to binary form and then place it in the core storage. The operation INPUT handles single items of data, the operation IN handles the input of one dimensional arrays.

The operation

INPUT (CARDS,A,B,C,...)

will read cards and transmit the numbers from them to the locations A,B,C, etc.

The operation

IN (CARDS,N,A)

will read N numbers from cards and transmit them in order to the locations A-1, A-2, etc., i.e., to the locations occupied by the array A.

If the data exists on BCD tape, written either by off-line equipment or by the 704 under control of the output operations, the following forms of the operations, are used:-

INPUT (TAPE M,A,B,C,...)
IN (TAPE M,N,A)

where M is an unsigned integer constant or an integer variable. M may take the values 1 to 10 inclusive.

INPUT is an exception to the rules regarding subscripting of operation arguments, in that no restriction is placed on any of its arguments other than that giving the tape number.

Rules for card punching:-

(1) Columns 1-72 may be punched.

(2) The numbers to be transmitted do not contain any blank characters, and must be separated from each other by single commas and/or one or more blank characters.

(3) The mode of the numbers is determined by the punching, and must be matched with the mode of the variables listed in the operation. Signs are indicated by a + or - sign preceding the number or exponent. It is not necessary to use the + sign. Fixed point integers are punched without a decimal point or exponent. They must not exceed 34359738367 in magnitude. Floating point numbers are punched with a decimal point and/or a decimal exponent preceded by an "E". If the decimal point does not appear it is assumed to be at the right hand end. If the character "E" does not appear the decimal point is assumed to be zero. Thus 12.345, +12.345E1, 1234.5E-2, and 12345E-3 are all equivalent representations of the same floating point number.

(4) If no digit appears between two commas, the card is read as if there were a zero between them.

(5) If the first punch on a card is a comma a zero is assumed to precede it. Similarly, if the last punch on a card is a comma a zero is assumed to follow it.

(6) A blank card is read as a single number zero.

The input conversion programme is adapted from UA DBC 1 and will consequently accept numbers with binary scale factors for conversion to fixed point binary quantities (see specification of UA DBC 1).

Notes:-

The numbers are read in order from left to right across the cards, and as many cards are read as are required to transmit all the numbers listed in the operation. If any numbers remain on the last card to be read, after the specified numbers have been transmitted, these will be lost: they are not read by the next input operation, which commences to read a new card.

(If the input is from tape, the remarks made about cards apply to records on the tape.)

If an end of file condition is met during input operations, a stop occurs with the octal number 70004 in the address field of the storage register. If the START button is pressed, the end of file condition will be ignored.

If in tape input operations an RTT failure occurs, two attempts will be made to read the record, and a subsequent failure will cause a stop with the octal number 70002 in the address field of the storage register. Press "START" to accept the record.

A2.2 Output.

As with input, there are two output operations. These are named OUTPUT and OUT, dealing with single items of output and one-dimensional arrays respectively.

The operation

OUTPUT (PRINT,F,A,B,C,...)

will print the numbers A,B,C etc. according to the format statement F. If it is desired to print Hollerith information only the operation is written OUTPUT (PRINT,F).

The operation

OUT (PRINT,F,N,A)

will print the N elements of the array A, namely A(1), A(2), etc.

If the output is required on cards or tape, the word PRINT should be replaced by the word CARDS or the words TAPE M, where M is an unsigned integer constant or an integer variable. M may take the values 1 to 10 inclusive.

If the operations OUTPUT or OUT are used to punch cards, the cards punched are entirely suitable for use with the input routines.

OUTPUT is an exception to the rules regarding subscripting of operation arguments in that no restriction is placed on any of its arguments other than that giving the tape number.

A2.3 Format.

To control the arrangement of the output, a format is used. Since the output conversion routine is adopted from UA BDC 1, formats may be written in SAP coding as specified for that routine (a restriction has been removed from that routine so that entirely Hollerith

formats and formats ending in Hollerith characters are now permitted), i.e., if the format symbol is F then the coding is

F TRA BLOCK

followed by BCD cards giving the required format. Such formats may be placed only where they are not reached by the sequencing of the programme. A suitable place is immediately before the word COMPILE. In addition, the symbol F must be defined as EXTERNAL.

However, the ABC translator includes a Format Generator for which the desired arrangement may be mapped on cards. This provides a simple and foolproof method of obtaining the required SAP formats.

## A2.4 Format Generation.

To generate a format, place the word "FORMAT" in the statement field of an ABC card. "FORMAT" cards may be placed anywhere in the programme, since the ABC translator will place the resultant format among the constants at the end of the programme. No extra words are allowed on a "FORMAT" card, except in the case specified below.

A "FORMAT" card must have a symbolic location so that it can be called for by name when required. It must be followed by 2 cards giving the lay-out for a line of printing. The columns 1-72 on the first card and the columns 1-48 on the second card each represent a type position on the printer.

Any Hollerith characters to be printed should be punched on these cards in the desired positions. The character "." should not be used as a Hollerith character, as it is used by the Format Generator to fix the position of the numbers to be printed. A format may consist entirely of Hollerith characters if desired.

Three basic types of conversion are possible with the output operations:-

| | |
|---|---|
| Integral binary to integral decimal | denoted by "I" |
| Floating binary to fixed decimal | " " "F" |
| Floating binary to floating decimal | " " "EXPNT" |

When printed, negative numbers are preceded by a "-" sign. Positive numbers remain unsigned.

To fix the position of a number for printing, a "." should be punched in the sign position of the maximum possible number to be printed (this will of course be determined by the programmer). A "." should also be punched in the desired position of the decimal point if the conversion is from floating binary, and finally one of "I", "F" of "E" should be punched in the last position to be occupied by the number. "E" must always be followed by "XPNT" in order to ensure that sufficient space is left for printing the exponent. "E" conversion produces the number in the form 0.xxx...

e.g., to print the numbers -9865 765.354 0.37995E-02
the card would be punched . I . . F . . EXPNT
in the desired positions.

A scale factor may be employed with the floating point numbers. If a digit is punched before the decimal point it is assumed to be a positive scale factor, and if after the decimal point a negative scale factor. Scale factors are restricted to one decimal digit. "E" conversion may only have positive scale factors. The scale factor is assumed to be zero if no value has been given, but once a value is given it will hold for all "F" and "E" conversion until a new one is given. Thus, if the above format were changed to

. I . .2 F .1. EXPNT
the numbers would be printed -9865 7.654 3.79953E-02

If the format specifies how to print only n numbers and the output routine is requested to print more than n numbers then additional lines with the identical lay-out will be printed, until all the numbers have been printed.

It may be desired to print a set of numbers with a different lay-out for each line. In this case the word "FORMAT" on the ABC card should be followed by a number saying how many different lines are to be specified, and the "FORMAT" card should be followed by this number of pairs of cards giving the different lay-outs. In this case also, if the number of numbers to be printed exceeds the number catered for in the format then the format will be repeated until all the numbers have been printed.

If a block format having the first lines of some special formats and all remaining lines of the same format is desired, the second card of the last lay-out pair should have something punched in at least one of the columns 49-72, which are normally blank.

The above instructions are given for printing. They are also applicable to BCD tape-writing and to card punching, although in the latter case columns 1-48 of the second card of the lay-out pair will always be blank.

## A2.5 The operation FORM.

It is possible, by the use of the operation FORM to alter a format specification during the operation of the object programme. The operation is written

FORM (CARDS,F)
or FORM (TAPE M,F) where F is the format symbol.

The FORM routine reads in the new UA BDC 1 type format (written in SAP code) from the standard BCD card form (i.e., col. 12 gives the number of BCD words on the card - it is blank if there are 10 words - while cols. 13-72 inclusive are used for the BCD words).

If more than one BCD card is used all cards except the last should have a punch in at least one of the cols. 1-6.

It is not essential to punch 'BCD' in cols. 8-11. The 'TRA BLOCK' instruction will be left unchanged at the beginning of the format.

It is important to realize that the new format must not be longer than the old, otherwise it may overwrite other useful information.

As with INPUT and IN, if an end of file condition is met while FORM is operating a stop occurs with the octal number 70004 in the address field of the storage register. If the START button is pressed the end of file condition will be ignored.

If an RTT failure occurs while FORM is reading from tape, two attempts will be made to read the record and a subsequent failure will cause a stop with the octal number 70003 in the address field of the storage register. If the START button is pressed the record will be accepted and computation will continue.

## Appendix 3

### A3. The library list of Functions and Operations.

The list of ABC library routines available for use at the present time is given below. Additions may be made at a later date.

#### A3.1 Functions.

There are no error stops in the function subroutines. When a function subroutine is given an argument for which it is unable to compute a correct value it returns a value whose modulus is the maximum $(2^{127}-1)$ or minimum (0) which may be stored in the 704. Details are given with the definitions below. Let x be the argument of functions. x is always a floating point number.

##### A3.11 Trigonometric functions. (Arguments in radians)

SIN,COS,TAN,COSEC,SEC,COT

Notes:- SIN If $x \geq 2^{80}$ zero is returned.
COS is computed as SIN $(x + \pi/2)$.
TAN If $|x| > 2^{26}\ \pi$ zero is returned.
COSEC,SEC and COT are computed as the reciprocals (using RECIP) of SIN,COS and TAN respectively.

##### A3.12 Inverse Trigonometric functions.

ASIN,ACOS,ATAN

Notes:- ASIN If $|x| > 1$ it is assumed to be 1. Function values lie in the 1st and 4th quadrants.
ACOS If $|x| > 1$ it is assumed to be 1. Function values lie in the 1st and 2nd quadrants.
ATAN Results take the sign of x.

##### A3.13 Exponential and Hyperbolic functions.

EXP,SINH,COSH,TANH,COSECH,SECH,COTH

Notes:- EXP If $x > 87.3$ the value $(2^{127}-1)$ is returned.
If $x < -87.3$ the value zero is returned.

SINH For arguments outside the range -87.3 to 87.3 the modulus of the value returned is $(2^{127}-1)$.

COSH As for SINH.

COSECH, SECH and COTH are computed as the reciprocals (using RECIP) of SINH, COSH and TANH respectively.

A3.14 Other functions.

LOG Natural logarithm, using $|x|$ as argument. Log(0) is $-(2^{127}-1)$.

SQRT Square root, using $|x|$ as argument.

GAMMA Gamma function. If x is a negative integer or if $x > 30$, the value $(2^{127}-1)$ is returned.

RECIP Reciprocal. If $|x| \leq 2^{-127}$ the modulus of the value returned is $(2^{127}-1)$.

INTEGP Integral part.

FRACTP Fractional part.

RANDOM Pseudo-random number generator. This generates a sequence of random numbers, equiprobable in 0,1 in the floating point mode. The starting pattern for the sequence generation is that of the modulus of the original value of the argument. For uses after the first, the argument ceases to have meaning, and has no effect.

A3.2 Operations.

For the purpose of definition, we use the following notation:-

| | |
|---|---|
| V,W,X,Y,Z | denote floating point variables. |
| I | denotes an integer variable. |
| A,B,C, | denote variables which may be either floating point or fixed point but must all be of the same type in any one operation. |
| N | denotes an integer count. |
| S | denotes a symbolic location. |

Any quantity which acquires a new value as the result of an operation will be underlined thus:- $\underline{X}$.

Operations may be divided into various classes:-

A3.21 Algebraic functional operations, i.e., functions of two algebraic variables.

| | |
|---|---|
| $\underline{Z}$ = PWR (X,I) | $Z = X^I$. |
| $\underline{Z}$ = POWER(X,Y) | $Z = (\text{mod } X)^Y$. |
| $\underline{Z}$ = MODULO(X,Y) | Z = X - Y * integral part of X/Y. If the values of X and Y differ so widely that X/Y is outside the range of numbers which may be computed in the 704, or if Y is zero, a stop will occur, with the octal number 70001 in the address field of the storage register. Pressing the START button causes the programme to continue, but an erroneous value will be accepted as the required value. |
| $\underline{A}$ = SIGN(B,C) | A = (sign C) * mod B. |
| $\underline{A}$ = MAX1(B,C) | A = the larger (algebraically) of B and C. |
| $\underline{A}$ = MIN1(B,C) | A = the smaller (algebraically) of B and C. |
| $\underline{A}$ = MAXM1(B,C) | A = the larger of mod B and mod C. |
| $\underline{A}$ = MINM1(B,C) | A = the smaller of mod B and mod C. |

A3.22 Operations on Arrays.

| | |
|---|---|
| $\underline{A}$ = MAX2(N,B) | A = the largest (algebraically) of the N elements of the array B. |
| $\underline{A}$ = MIN2(N,B) | A = the smallest (algebraically) of the N elements of the array B. |

| | |
|---|---|
| A = MAXM2(N,B) | A = the largest of the moduli of the N elements of the array B. |
| A = MINM2(N,B) | A = the smallest of the moduli of the N elements of the array B. |
| Z = SUMSQS(N,X) | A = the sum of the squares of the N elements of the array X. |
| Z = RADIUS(N,X) | A = the square root of the sum of the squares of the N elements of the array X. |
| SORTA(N,B) | The N elements of the array B are sorted in ascending order. The smallest becomes B(1). |
| SORTD(N,B) | The N elements of the array B are sorted in descending order. The largest becomes B(1). |
| SORTPA(N,A,B) | The N elements of the array A and the N elements of the array B are considered to be related pairs of values. The elements of the array A are sorted in ascending order, and the elements of the array B are sorted to correspond, so that each element of A retains its original partner in B. |
| SORTPD(N,A,B) | Parallel sorting, as in SORTPA, but the array A is sorted in descending order. |
| SET1(N,A,B) | The N elements of the array A are set to the value B. |
| EXCHNG(N,A,B) | The N elements of the array A are interchanged with the N elements of the array B. |
| TRANSF(N,A,B) | The N elements of the array B are set to the same values as the N elements of the array A. |
| PUNCH(N,A,B) | The N elements of the array A are punched onto standard binary cards (with word count, loading address and checksum) suitable for reloading into the machine. B is the name of the array into which the elements will be read on reloading. If B = A the operation must be written PUNCH(N,A,A). |

| | |
|---|---|
| MAX4(N,A,I,B) | A = the largest (algebraically) of the N elements of the array B.<br>I = the subscript of this element. |
| MIN4(N,A,I,B) | A = the smallest (algebraically) of the N elements of the array B.<br>I = the subscript of this element. |
| MAXM4(N,A,I,B) | A = the largest of the moduli of the N elements of the array B.<br>I = the subscript of this element. |
| MINM4(N,A,I,B) | A = the smallest of the moduli of the N elements of the array B.<br>I = the subscript of this element. |
| POLY1(N,X,Y,Z) | Computation of polynomial of order N. Z is an array of order (N + 1).<br>X = Z(1) + Z(2) * Y + .....+ Z(N+1)*. |

A3.23 Operations on collections.

| | |
|---|---|
| MAX3(N,A,B,C,..) | A = the largest (algebraically) of the N quantities B,C,.... |
| MIN3(N,A,B,C,..) | A = the smallest (algebraically) of the N quantities B,C,.... |
| MAXM3(N,A,B,C,..) | A = the largest of the moduli of the N quantities B,C,.... |
| MINM3(N,A,B,C,..) | A = the smallest of the moduli of the N quantities B,C,.... |
| POLY2(N,Z,X,V,W,...Y) | Z = polynomial of degree N, with argument X and coefficients V,X,...Y.<br>Z = V+W*X+.....+Y*X^N. |
| SET2(N,A,B,...,C) | The N variables A,B,... are set to the value C. |

Note:-

All function arguments and those arguments of operations which are single variables and are not underlined, may be constants if desired. Indeed, the integer count N will in most cases be a constant. If it is a variable it may take any integer value less than or equal to the number of arguments for which the count is used. For example, the operation MAX3(N,A,B,C,D,E) may be used for N = 4 or less. In the case of arrays N must be less than or equal to the dimension of the array. In the operation SET2, N is used to select the value to which the preceding N variables are to be set. Thus in normal use the operation SET2 has a total of (N+2) arguments.

A3.24 Mathematical processes.

(a) Quadrature. 5-point Gaussian integration. Let the integrand be calculated by a closed subroutine or SECTION, and let the name of the section be S. It calculates X = F(X). the integrand, taking its argument as X and replacing it by the corresponding function value. Then the operation

GAUSS(X,Y,Z,S)

performs the definite integration between the limits Y and Z. Note that S is a symbolic location argument and must therefore be defined as EXTERNAL.

Example. To find $\int_y^z e^{-x}.\cos x^2 dx$

(1) Definitions.

```
ABC EXTERNAL S
ABC FUNCTIONS EXP COS
ABC OPERATION GAUSS
```

(2) Calculation of the integral.

```
ABC GAUSS(X,Y,Z,S)
```

(3) Calculation of the integrand, (to be placed outside the sequencing of the main programme)

```
S  ABC SECTION S
   ABC X = (EXP-X)COS X*X
   ABC END OF SECTION S
```

The variable X has the value of the integral after the operation.

(b) Finding a zero of a function. If a function is defined by a closed subroutine or section as in the previous example, and Y is an estimate of the root required, then the operation for improving the estimate of the root is

ROOT(X,Y,W,V,S)

This will place the improved estimate of the root of the function in X. S is the name of the section to calculate the function. W is the step interval. If the initial estimate of the root is Y, then the function is calculated at Y and Y+W. Thereafter the argument X is stepped towards the root at interval W until the root is bracketed. Then iterative linear interpolation is used until the value of the function falls below V in magnitude, or until two successive iterates agree.

Example. To find a root of the equation $x^8-2x-5 = 0$, using $x = 2$ as a first approximation.

(1) Definitions.

```
ABC EXTERNAL S
ABC OPERATION ROOT
```

(2) Refinement of the root.

```
ABC ROOT (X,2.0,0.1,0.000001,S)
```

(3) Calculation of the function.

```
S  ABC SECTION S
   ABC X = X(X*X-2)-5
   ABC END OF SECTION S
```

We have chosen the stop to be 0.01 and the convergence number to be 0.000001. The estimate must be written as 2.0, since a floating point constant is required.

(c) To advance by one step the integration of n simultaneous ordinary differential equations of the first order,

$$\frac{dy}{dx}i = f_1(y_1, y_2, \ldots y_n), \quad i = 1, 2, \ldots n$$

i.e., to calculate the values of the variables $y_1$ when x is increased by h.

A form of the Runge-Kutta method is used, and the operation is written

RUKU(Y,Z,W,V,N,S)

where V is the interval of integration.
N is the number of equations to be solved.
Y is the name of the array of order N containing the starting values $y_1, y_2, \ldots y_n$. After the operation, Y will contain the new values of $y_1, y_2, \ldots y_n$.
S is the location of an auxiliary routine which must be supplied by the programmer to calculate the values of the functions $f_1, f_2, \ldots f_n$ (finding the current values of the arguments $y_1, y_2, \ldots y_n$ in the array Y) and place them in the array Z, which must of course be of order N.
W is the name of an array of order N which is used by the routine to hold rounding-off errors, which are taken into account on the next step in order to prevent their rapid accumulation. The array W must be cleared at the beginning of each integration and at the beginning of each range in which V assumes a different value.

The truncation error in one step is $O(V^5)$. For a small set of well-behaved equations it is approximately $10^{-2}V^5$.

If V is not an exact binary number and x appears explicitly in the functions $f_i$, there is a definite numerical advantage to be gained by treating x as another dependent variable and generating it by integration of the equation $x' = 1$.

If V is an exact binary number and x appears explicitly in the functions $f_i$, and is not generated as a dependent variable, the values x, $x+\frac{1}{2}V$, $x+\frac{1}{2}V$, $x+V$ must be supplied to the auxiliary routine at the four stages of each step.

An example of a complete programme using RUKU is given below:-

Given the equations $y' = -z$ and $z' = y$ with the initial conditions $y(0) = 1$ and $z(0) = 0$, to print out the values of x, y, z at $\overline{x} = 0(.01)1.0$ correct to 7 decimal places. (It will be noted that the required interval is small enough to ensure the required accuracy.)

```
      ABC OPERATIONS RUKU OUT SET1
      ABC DIMENSION Y 3
      ABC DIMENSION Z 3
      ABC DIMENSION W 3
      ABC EXTERNAL S
C     ABC Y(1) = 0
      ABC Y(2) = 1.
      ABC Y(3) = 0
      ABC SET1(3,W,0)
      ABC OUT (PRINT,B,3,Y)
B     ABC FORMAT
. .     F     . .     F     . .     F
        (blank card)
```

```
      ABC DO SECTION D I = 1,1,100
      ABC STOP
D     ABC SECTION D
      ABC RUKU(Y,Z,W,.01,3,S)
      ABC OUT(PRINT,B,3,Y)
      ABC END OF SECTION D
S     ABC SECTION S
      ABC Z(1) = 1.
      ABC Z(2) = - Y(3)
      ABC Z(3) = Y(2)
      ABC END OF SECTION S
      ABC COMPILE
      END C
```

A 3.25 Binary input-output operations.

There is a set of four operations which will transfer binary information to or from drums, tapes or cards. The operations READ and WRITE will transfer collections of single items, while the operations READA and WRITEA will transfer arrays.

READ(CARDS,A,B,...) will read one card and transfer not more than 24 separate binary items to the locations A,B,...

READA(CARDS,N,A) will read N items from binary cards and transfer them to the array A. (More than one card will be read if N is greater than 24)

READ(DRUM M,L,A,B,...) will read binary information from drum M, commencing at drum location L, and store it in locations A,B,...

READA(DRUM M,L,N,A) will read N consecutive words from drum M, commencing at drum location L, and store them in the array A.

No checking is done by these operations when reading from cards or drums.

READ(TAPE M,A,B,...) will read one record of binary information from tape M and store the words in the locations A,B,... A READ(TAPE M) may be given without a list of locations, in which case the programme will skip over a record or end of file mark. If the list is longer than the record, the programme will stop with the octal number 70005 in the address field of the storage register. The object programme redundancy checks the tape reading. If a record fails twice the programme stops with the octal number 70006 in the address field of the storage register. Pressing the START

button causes the information read on the second attempt to be accepted. If an end of file mark is encountered the object programme will stop with the octal number 70007 in the address field of the storage register. Pressing the START button causes the programme to read the next record.

READA(TAPE M,N,A) will read N items from tape M and store them in the array A. The stops in the object programme are effectively the same as for READ, except that the numbers in the storage register are 70010,70011 and 70012 respectively.

The operations WRITE and WRITEA are used in the same manner as READ and READA to transfer information to cards, drums and tapes. No checking is done by WRITE and WRITEA.

## Appendix 4.

### A4. Operating notes for the ABC system.

**To write an ABC system tape.**

Set a machine tape to logical tape 1.
Put sense switch 6 up.
Ready the card deck AB C 000-881 in the card reader.
Press CLEAR and LOAD CARDS,

The ABC system will then be copied from the cards onto tape 1, which will be rewound, and the 704 will stop at location 0.

**To copy an ABC system tape onto another tape.**

Set the existing ABC system tape to tape 9.
Set a machine tape to tape 1.
Put sense switch 6 down.
Ready the card deck AB CC 00-76 in the card reader.
Press CLEAR and LOAD CARDS.

The ABC system will then be copied from tape 9 to tape 1, both tapes will be rewound, and the programme will end with a LOAD CARDS simulator.

**To compile a programme with the ABC system.**

Load the ABC system tape as tape 1.
If off-line printing of the assembly is required, set a machine tape to logical tape 2.
Set a machine tape to logical tape 3.
If the symbolic deck to be compiled has been written on tape, load this tape as logical tape 4. If the symbolic deck is to be read from cards set a machine tape to logical tape 4.

Sense switches 1-5 are used for the SAP assembly, therefore their settings are as follows-

| | |
|---|---|
| 1 Up and 2 Up | Input to both passes of the assembly programme is always from tape 3. |
| 3 Up | Suppress on-line printing. |
| 3 Down | Output is printed on-line. |

| | |
|---|---|
| 4 Up | Any on-line printing is single spaced. |
| 4 Down | Any on-line printing is double spaced. |
| 5 Up | Logical tape 2 is written during the second pass of the assembly programme, in order to permit later off-line printing. |
| 5 Down | Suppress preparation of tape 2. |

The setting of sense switch 6 is not relevant to the assembly programme since a separate library tape is not permitted by the ABC system. It is therefore used to distinguish between input from cards or tape to the ABC system.

| | |
|---|---|
| 6 Up | Input is from tape 4. |
| 6 Down | Input is from cards read on-line. |

Control panel requirements are as for UA SAP.

Ready the following card deck in the card reader and press CLEAR and LOAD CARDS.

1) Primary control card ABC PRI.

2) Secondary control card ABC SEC.

3) Deck to be compiled (ending in COMPILE and END cards) if it is to be read on-line.

4) Symbol table from previous assembly (if any).

5) Correction-transfer card for SAP1.
ABC NST if symbol table is not required.
ABC ST if symbol table is required.

6) Two blank cards.

Items 2-5 are repeated if more than one compilation is to be done. If several programmes are to be compiled successively they may be written on the same input tape, the last one being followed by an End of File mark.

After all compilation and assembly is completed the ABC programme ends with a LOAD CARDS simulator.

The deck of cards produced by the ABC system consists of the following-

1) The ABC upper one-card loader.
   This loader will accept absolute binary cards only. The checksum stop is at location -2. If 9L2 is punched the checksum will be ignored.

2) Any library routines requested.

3) The cards punched by the assembly programme.

If the symbol table is not punched by the assembly programme the above deck is completely ready for loading into the 704 as an operational programme. If the symbol table is punched, it and the two blank cards following it must be removed from the deck before it is loaded.

## Appendix 5.

### A5. Extending the ABC library.

Library routines are stored on the system tape in the form of closed relocatable subroutines (starting at origin 0) - the system tape being copied originally from relocatable cards.

A routine may use other library routines as subroutines. If the routine is a function routine the argument of the function is found in the AC when the routine is entered. Index register 4 is used to enter subroutines, as in normal practice. If the routine is an operation routine, the arrangement of the arguments is best explained by an example -

MAX3(4,A,-B(I),-C,D(I)) is translated by the ABC programme to

```
LXA I,1
CLA D,1
CLS C
CLS B,1
CLA A
CLA Location of 4
TSX MAX3,4
```

Thus index register 4 can be used for collecting the various arguments of an operation.

Index registers 1 and 2 must be preserved by library routines. It should be borne in mind that the arguments may be indexed and may be positive or negative, so the entire instruction must be studied by a library routine when collecting the arguments. A convention that if an operation has more than two arguments the place where the result is to be stored is inserted as an additional argument has been followed in the set of library routines issued with the ABC system.

Each library routine has a serial number, which is used by the library-writing routine and the ABC translator if the routine is called for as a subroutine of another. Most library routines have names (a symbol of 6 or less characters), which are used by the translator when searching

for a routine which has been defined as a FUNCTION or OPERATION and not as an EXTERNAL. Those library routines which have no name are used only as subroutines of other routines and have no use on their own.

Each library routine has associated with it a relocatable index card which is assembled in the following manner -

```
ORG 0
BCD 00NAME
HTR a,0,b
HTR Q,0,d  or  MZE c,0,d     } 9 possible references
HTR e,0,f                    } to other routines may fill
                               the rest of the card.
```

i.e. Name at righthand end of BCD word, preceded by zeros.
- a = number of locations used by the routine.
- b = serial number of the routine.
- f = serial number of subroutine referred to.
- d = address referred to in the subroutine.
- e = position of reference.
- c = second position of reference (if it exists). Use MZE for a second reference.

A library routine must consist of only one continuous block of information completely filling the number of locations specified (i.e. no BSS or BES pseudo-ops are allowed).

Library routines must be arranged on the system tape in such a way that a routine using another routine as a subroutine appears before it on the tape. The serial number order of the routines has no significance in this respect - serial numbers were allocated to routines as they were written. In the card deck AB C 000-881, the columns 76 and 77 are used to hold the library serial number of the routine to which the card belongs. These columns are blank on cards which are not part of the library. A list of the present order of routines is appended, and an updated list can be obtained by using the library-printing programme AB CL PR 00-30 if routines are added to the library. This list treats each library routine in the order in which they appear on the tape, printing the serial number, name, and the serial numbers of the 9 possible subroutines referred to.

The library is written on the system tape in the order -

All index cards.
A blank card (AB C 427 in the current AB C deck).
Index card plus programme cards for each routine in turn.
End of file mark (produced by a blank card - AB C 732 in the current AB C deck).

Thus two copies of each index card are required. The same library writing programme is incorporated in the decks AB C 000-881 and AB CC 00-76, AB CC 37 = AB C 427 and AB CC 38 = AB C 732. If sense switch 6 is up the entire library may be copied from cards to tape 1. If sense switch 6 is down the library will be copied from tape 9 to tape 1, with modifications indicated by control cards inserted before the two blank cards mentioned above.

If a routine is to be removed from the library, a control card should be punched with its serial number in 9LA, and the 9RS bit punched. If a routine is to be added to the library, 9LA of the control card should be punched with the serial number of the routine it is to follow, and the 9R1 bit should be punched.

Thus if an amended version is to replace a routine already on the tape the control card should have its serial number in 9LA, and 9RS and 9R1 should be punched. If more than one routine is to be added in the same place, one control card will suffice, if it is punched in 9R 1,2,3... - one column for each routine to be added.

When considering the serial number of a routine to be followed by an addition it should be borne in mind that the programme regards as the previous serial number that of the last routine processed - therefore it may be the last one copied from tape, the last one removed from tape, or the last one added to the tape. If additions are to be made at the beginning of the tape, leave the serial number blank on the control card. If several additions are to be made, the order of the routines with which they are connected must of course be considered when collecting the control cards.

When using AB CC to alter the library the order of the cards is as follows -

AB CC 00-36
Control card
Index card(s) of addition(s) } repeats if more than one control card.
AB CC 37
Control card
Index card plus programme cards for each addition } repeats if more than one control card.
AB CC 38
AB CC 39-76

Thus two copies of each control card are required in addition to two copies of each index card.

The routines with serial numbers 2,3,4,33,47 and 48 are single location routines used as COMMON storage by the other routines - they are not necessarily consecutive locations.

The two library routines INP1 (serial 68) and RTX (serial 69) are versions of the routines NY INP1 and UA CSH2 respectively, which may be used with their normal SAP calling sequences, apart from the fact that the end of file or error returns have been incorporated in the routines and therefore the calling sequences only use 2 locations instead of 3.

To print the library list.

Set the ABC system tape to logical tape 1.
Ready the card deck AB CL PR oo-30 in the card reader.
Press CLEAR and LOAD CARDS.
The programme will then print the library list (Share 1 printer panel), rewind the system tape and stop at location 253.

The ABC library in order as on the system tape.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 84 | PWR | 3 | 3 | 4 | 57 | 0 | 0 | 0 | 0 | 0 |
| 83 | RUKU | 2 | 3 | 4 | 4 | 33 | 33 | 33 | 0 | 0 |
| 82 | POWER | 33 | 33 | 43 | 1 | 0 | 0 | 0 | 0 | 0 |
| 80 | PUNCH | 2 | 3 | 4 | 4 | 33 | 33 | 0 | 0 | 0 |
| 78 | WRITEA | 79 | 79 | 79 | 79 | 79 | 0 | 0 | 0 | 0 |
| 77 | READA | 79 | 79 | 79 | 79 | 79 | 0 | 0 | 0 | 0 |
| 79 | 0 | 60 | 73 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 75 | WRITE | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 0 |
| 74 | READ | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 | 76 |
| 76 | 0 | 60 | 73 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 72 | GAMMA | 5 | 1 | 40 | 47 | 47 | 33 | 33 | 33 | 0 |
| 67 | FORM | 66 | 69 | 69 | 69 | 0 | 0 | 0 | 0 | 0 |
| 65 | IN | 4 | 66 | 68 | 68 | 68 | 68 | 0 | 0 | 0 |
| 64 | INPUT | 66 | 68 | 63 | 63 | 63 | 68 | 63 | 63 | 0 |
| 68 | INP1 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 69 | 69 |
| 66 | 0 | 60 | 69 | 69 | 0 | 0 | 0 | 0 | 0 | 0 |
| 69 | RTX | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 0 | 0 |
| 62 | OUT | 59 | 61 | 61 | 61 | 0 | 0 | 0 | 0 | 0 |
| 58 | OUTPUT | 59 | 59 | 63 | 63 | 63 | 63 | 63 | 0 | 0 |
| 63 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 59 | PRINT | 60 | 61 | 61 | 61 | 61 | 61 | 0 | 0 | 0 |
| 60 | CARDS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 61 | BLOCK | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 56 | COSECH | 49 | 57 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 55 | SECH | 50 | 57 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 54 | COTH | 41 | 57 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 53 | COSEC | 5 | 57 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 52 | SEC | 6 | 57 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 51 | COT | 44 | 57 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 57 | RECIP | 2 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 |
| 50 | COSH | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 49 | 0 |
| 49 | SINH | 2 | 2 | 2 | 2 | 3 | 3 | 4 | 33 | 33 |
| 46 | ASIN | 4 | 4 | 45 | 45 | 45 | 45 | 45 | 0 | 0 |
| 45 | ACOS | 40 | 4 | 4 | 4 | 33 | 47 | 47 | 48 | 0 |
| 44 | TAN | 2 | 2 | 3 | 3 | 3 | 3 | 4 | 0 | 0 |
| 43 | LOG | 2 | 2 | 2 | 2 | 3 | 4 | 4 | 0 | 0 |
| 42 | ATAN | 2 | 3 | 3 | 3 | 4 | 33 | 0 | 0 | 0 |
| 41 | TANH | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 0 |
| 39 | RADIUS | 40 | 4 | 4 | 33 | 33 | 0 | 0 | 0 | 0 |
| 40 | SQRT | 2 | 3 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| 38 | SUMSQS | 2 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 37 | EXCHNG | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 36 | TRANSF | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 34 | SET1 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 32 | SORTPD | 2 | 2 | 3 | 3 | 4 | 33 | 0 | 0 | 0 |
| 31 | SORTPA | 2 | 2 | 3 | 3 | 4 | 33 | 0 | 0 | 0 |
| 30 | SORTD | 2 | 2 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| 29 | SORTA | 2 | 2 | 3 | 3 | 0 | 0 | 0 | 0 | 0 |
| 28 | FRACTP | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 27 | INTEGP | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 25 | POLY1 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 24 | MODULO | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 0 | 0 |
| 23 | SIGN | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 22 | MINM4 | 2 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 21 | MAXM4 | 2 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 20 | MIN4 | 2 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 19 | MAX4 | 2 | 3 | 3 | 0 | 0 | 0 | 0 | 0 | 0 |
| 14 | MINM2 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 13 | MAXM2 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 12 | MIN2 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 11 | MAX2 | 2 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | COS | 3 | 5 | 5 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | SIN | 2 | 2 | 2 | 2 | 3 | 3 | 0 | 0 | 0 |
| 1 | EXP | 2 | 2 | 2 | 3 | 4 | 4 | 4 | 0 | 0 |
| 2 | TAPE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | MAX1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | MIN1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | MAXM1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | MINM1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 15 | MAX3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 16 | MIN3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 17 | MAXM3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 18 | MINM3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 26 | POLY2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 33 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 35 | SET2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 47 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 48 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 70 | RANDOM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 71 | GAUSS | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 73 | DRUM | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 81 | ROOT | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Programme stops.

AB C 000-881.

7 Checksum failure. Reload the last 4 cards read and press LOAD CARDS.

8 Checksum failure. Reload the last 6 cards read and press LOAD CARDS.

15 Checksum failure. Reload the last 3 cards read and press LOAD CARDS.

24 Tape loader copied incorrectly onto tape. Press START to try again.

53 Card checksum error. Press START to accept.

66 Checksum wrong on tape. Press START to try again.

100 Checksum wrong on tape. Press START to try again.

106 Machine failure or card deck in error.

107 Machine failure or card deck in error.

212 Library card checksum failure. Press START to continue (checksum is corrected on tape).

AB CC 00-76.

17 Tape checksum error. Press START to try again.

212 Library card checksum failure. Press START to continue (checksum is corrected on tape).

240 Error - end of file while reading library programme from tape 9. Machine failure or tape 9 wrong.

313 Error - end of file on tape 9 before AB CC 37 and 38 are read. Alterations probably in wrong order.

327 Checksum failure while reading library programme from tape 9. Serial number is in MQ address, loading address in MQ decrement, checksum in AC. Press START to continue.

335 Checksum failure while reading library index card from tape 9. Serial is in MQ address, checksum in AC. Press START to continue.

444 Blank card image on tape 9 before AB CC 37 is read. Alterations probably in wrong order.

Programme stops while the ABC system is operating.

The stops in the assembly part of the system are the usual ones for UA SAP.

66 Checksum error on tape loader. Press LOAD TAPE to try again.

77 Checksum error during reading of system tape. Press START to try again.

106 Checksum error during reading of system tape. Press START to try again.

237 Illegal punch on symbolic card. Load corrected card and press START to continue.

773 Number of ( brackets on a card is not equal to the number of ) brackets.

6512 Function or operation name without argument.

13016 }
13021 } Errors in format lay-out.
13123 }
13175 }

15724 Checksum failure while reading library records. Press START to try again. If repeated failures, system tape is faulty and should be rewritten.

Any other stops which occur will be due to machine faults or incorrect notation on ABC cards. Two courses of action are possible if such a stop occurs -

1) Transfer manually to location 13623. This will cause the card which is giving trouble to be ignored by the ABC system, which will proceed to translate the next card. After the assembly process is completed, a study of the listing will show which card was in error, and it should be possible to correct the binary deck by patching.

2) Abandon the translation process and print out the contents of tape 3 either off-line or with an off-line simulator. Tape 3 contains the symbolic deck which the ABC system has prepared for input to the assembly programme, and the last REM card on the list will show which card is in error.

Addendum.

Modes of certain operations.

Operations such as MAX1(A,B) may be used with the arguments either in fixed or floating point mode (but not both at once).

When such an operation has fixed point arguments it will have a fixed point result and this must be signified in the notation by placing a $ symbol before the name of the operation. For example, if B and C are fixed point variables we must write $MAX1(B,C) whenever we wish to use this (and other similar operations) in the fixed point mode.