ZASTOSOWANIA MATEMATYKI
APPLICATIONES MATHEMATICAE
IX, 3 (1968)

Krystyna SOCHACZ and J. SZCZEPKOWICZ (Wrocław)

A DESCRIPTION OF THE AS-LANGUAGE FOR THE ELLIOTT 803 COMPUTER

0. Summary.

A simple but powerful programming language is described, which has been designed and implemented by the authors on the National Elliott 803 computer to assist in preparing readable machine-coded programmes for that machine. This language, called Addressing Symbolism (AS), contains the whole Elliott TI code and allows the use of symbolic addresses and independent levels of nomenclature. A precise definition of the syntax of the language and some important features of the translator are given. Some knowledge on the NE 803 computer is assumed. but the main features of the language may be well understood even without this knowledge.

1. Introduction.

The programing languages used up to now at Wrocław University to programme the Elliott 803 computer were the Elliott Algol, the MARK III Autocode and the so called TI code. The latter was designed for preparing machine coded programmes by hand. In the Elliott Algol and MARK III languages there are also facilities for writing a piece of programme in machine code. Thus there are, theoretically, three languages which might be used to make a machine coded programme. It follows from experience that no one of these possibilities is fully satisfactory when a fairly big machine coded programme is to be written. As a matter of fact, only the TI code satisfies the demand for an efficient object programme with the respect to any feature the programmer may think about. Making a big TI coded programme, however, is a very unpleasant chore, since in TI there are only octal/decimal, octal and decimal notations, and a few facilities such as relative addresses and partitioning the programme into blocks. Alphanumeric data may also be handled, but these should be carefully divided into groups of six

characters. All the address calculations and assignments must be done by hand. This is, evidently, a highly clerical task; each experienced programmer writes his machine codings using an "ad hoc" symbolic notation of his own, and then replaces all symbols by suitable addresses. It is observed that most of the errors are introduced at the latter stage. The NE 803 software supplied by the manufacturer contains no item to mechanize this stage, and this was the reason for constructing a new language called AS (Addressing Symbolism). The main features of the AS programming system may be summarized as follows:

(a) The TI code is a proper subset of the AS language. This feature is extremely valuable, since the numerous Elliott TI subroutines can be used without any change, including relative addressed binary tapes (TICRB tapes).

(b) Any number of independent levels of nomenclature may be used in the source programme. This means that a closed AS subroutine may be inserted into any AS programme with no restrictions posed on the identifiers used in the rest of the programme.

(c) Alphanumeric data may be handled with the same ease as in the MARK III Autocode.

(d) There are no identifiers reserved in the AS language.

(e) No jump-decode tables are inserted by the translator into the object programme with the effect that the latter is as efficient as it was intended in the source programme.

(f) The AS translator reduces to a minimum manipulating the machine control keyboard. The output of information useful in debugging the programme and correcting it after translation is optional.

(g) The AS translator provides an extensive checking of the source programme and precise error indications. A considerable number of errors can be corrected without retranslating the programme from the beginning.

(h) Unless specifically demanded, all data read so far by the translator and the translator itself are automatically protected against overwriting due to programmer's error.

(i) The AS translator is not much slower than the TI translator. The latter reads about 3 machine words per second; a real AS programme of 1100 words containing 1500 symbolic addresses and over 300 declared names has been translated in less than 520 seconds. The time quoted includes sorting various name lists and final addressing after reading the programme in.

(j) The AS translator occupies less than 900 words together with its working locations and thus can be practically used even on machines with 2048 word store. It is easy to arrange the programme in such a way that the translator is overwritten at run time of the translated programme.

2. The syntax and semantics of the AS language. The methodology of the translator.

The alphabet of the AS language consists of the Elliott Telecode characters. To specify the syntax of the language we will apply the method of Iverson [1]. In the syntax table below the following conventions are used:

(a) The symbols of defined syntactic units are two-digit numbers. They are used also for reference purposes.

(b) A § is used to denote the syntactic unit being defined; a recursive definition contains therefore a §.

(c) The allowable forms of a syntactic unit are separated by vertical lines.

(d) Any other symbol, not defined but occurring in a definition, represents itself.

(e) Two-letter symbols are used to denote page layout characters and blanks, namely

bl for *blank*,
sp for *space*,
cr for *carriage return*,
lf for *line feed*.

(f) The characters *bl cr sp* may be used arbitrarily in all those syntactic units whose definitions do not contain these characters explicitly or implicitly.

The syntax table is followed by suitably arranged comments to describe the semantics of the language and relevant information on the translator.

The syntax table of the AS language

NAME	SYM-BOL	DEFINITION
empty	Ø	
letter	10	**A**|**B**|**C**|···|**X**|**Y**|**Z**
octal digit	11	**0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**
digit	12	**0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**
teleprinter character	13	10|12|*|**$**|=|'|,|:|−|.|%|(|)|+|**?**|/|∂|£|*sp*|*cr*|*lf*
alphanumeric character	14	13|*bl*
identifier	15	10|§ 10|§ 12
unsigned integer	16	Ø|12|§ 12
signed integer	17	+16|− 16
absolute address	18	16|17

NAME	SYM-BOL	DEFINITION
relative address	19	18 , 16
integer constant	20	17|17 , 16
symbolic address	21	15|15 17
preset parameter	22	16'|+ 16'|19'|§ §
address	23	18|22 18|19|22 19|21|22 21
function part	24	11 11
B-line	25	:|/
instruction	26	Ø|24 23
instruction pair	27	26 25 26
trigger	28	27 (
octal group	29	**8** 11 11 11 11 11 11 11 11 11 11 11 11 11 11
alphanumeric group	30	Ł 14 14 14 14 14 14
comment	31	13|§ 13
title	32	$ 31 *bl*
copied group	33	Ø|= 31 *bl*
fixed-point fraction	34	17 . 16
floating-point number	35	34 / 17|34 / 16
label	36	Ø|15)
word	37	20|27|29|30|32|34|35|%₀
line	38	33 36 37 *lf*|28
temporary directory	39	20 $ *lf*
group of lines	40	38|39 §|§ §
maximum index	41	Ø|(16)
label list	42	15|§ , 15
variable list	43	15 41|§ , 15 41
label declaration	44	Ø|**LABELS:** 42 *lf*
variable declaration	45	Ø|**VARIABLES:** 43 *lf*
declaration	46	44|45|§ §
local declaration	47	Ø|46 **BEGIN**: *lf*
end of block	48	* *lf*|)
block	49	47 40 48
group of block	50	49|§ 49
working space declaration	51	Ø|**WORKSPACE:** 16 — 16 *lf*
list of block addresses	52	33 36 + 16 *lf*|§ §
directory	53	33 *∂ lf* 51 52 46 48
programme	54	53 50|53 50 **END:** *lf*|§ ⟨TI coded blocks⟩

Remark. For typographical reasons the teleprinter character @ has been changed to *∂*.

The symbols not discussed in the below references are either self-evident or of secondary importance. In any case the reader is referred to [2] for further information.

Ref. 13. A teleprinter character is defined to be one that moves the carriage or paper. Shifts are not mentioned here as they may be used arbitrarily by the typists provided that the printed form of the programme is correct when reproduced automatically on the teleprinter.

Ref. 15. The definition of the identifier conforms to the corresponding ALGOL definition, but identifiers longer than six characters are automatically truncated to six leftmost characters. Spaces can be used arbitrarily within the identifiers. Thus the identifiers

FILM REWIND
FILM REW
FILMRE

are all assumed to be identical while the identifiers

INFEAS
INF1 S
INF3A

are all distinct.

Ref. 19. Relative addresses are interpreted just as in the TI code.

Ref. 21. An address of the form

IDENT

denotes "the absolute address assigned to the identifier **IDENT**" while the address

IDENT+5

denotes "the absolute address assigned to the identifier **IDENT** plus 5". The form **IDENT-5** is also allowable and has an analogous interpretation. The assignment of addresses is done either automatically by the translator, or is given explicitly by the programmer, or both. See also ref. 52.

Ref. 24. A function part consists of two octal digits. In the authors' opinion, with a systematic instruction code such as the Elliott's one, there is no need for a symbolic function part.

Ref. 25, 26, 27. An instruction pair may be also punched in some abbreviated form, e.g.

: (representing a zero word)
40 STOP: (representing **40 STOP : 00 0**)

Ref. 28. The left bracket following an instruction pair causes the translator to transfer control to the working location where the pair has been stored. If the trigger does not contain an effective jump in-

struction, the machine stops. Triggers should be used with care, as the programme may be far from being complete when the trigger is being obeyed (see ref. 54 and 3.6).

Ref. 30. This is a relic of the TI code no more useful in AS programmes.

Ref. 32. A title is a word starting with a $ sign and ending with a *bl.* Thus the title may contain any teleprinter characters except of blanks. The characters are stored in consecutive locations, six in one, together with an implicit end symbol, and are printed by means of the TU12 subroutine. The following example shows how titles are stored: the characters

$ *cr lf* **AX+BY+CZ = 15** *bl cr lf*

produce the same machine words as the characters

80353641701342 *cr lf*
80711343720601 *cr lf*
−21 *cr lf*

Ref. 33. A copied group looks like a title with the starting $ replaced by a = . This group is copied on the output (punch 1) and is completely ignored by the translator in all other respects.

Ref. 36. Any word can be labelled with an identifier declared as label (see ref. 44). A right bracket separates the identifier from the word.

Ref. 37. Although a title occupies in general more than one machine word, it is also treated as a word. A word containing $^0/_0$ only is used to skip a location. This is of value when modifying a programme already in store.

Ref. 38. A line is defined to be a word followed by a *lf.* Thus the blanks which terminate a title must also be folowed by a *lf.* Each word may be preceded by a copied group and a label, generally in that sequence. The other sequence also may be used, but without erasing any information on the line read so far (see [2]).

Ref. 39. A temporary directory is used to indicate, where the information following it should be placed. Its effects are just as those described in [2]. A temporary directory has also the effect of "switching off" the store protection mechanism mentioned in the Introduction under (h) thus providing a facility to modify the programme already in store.

Ref. 41. The maximum index is used in the variable declaration to state that an array of store locations should be reserved for the specified identifier. If, for instance, the symbols

FUNVAL(150)

occur in the variable declaration then 151 consecutive locations are reserved for the identifier **FUNVAL** to store the values of 151 indexed varables

FUNVAL, FUNVAL+1, FUNVAL+2, ..., FUNVAL+150

Ref. 42, 43, 44, 45, 46. An example of the label declaration is

LABELS: READ, PRINT, L1, LAB, STOP, END, COLFAC

and an example of the variable declaration is

VARIABLES: I, J, COL, FUNVAL(150), ARG(150), STACK

Those two types of declaration may occur in any sequence and may be mixed in any mode. At this point it should be emphasised that no line of the programme (except of those containing titles and copied groups) may contain more than 70 printable characters; thus as many **LABELS:** and **VARIABLES:** should be used as many lines are required to declare all the identifiers wanted at a particular level of nomenclature (see ref. 47).

The identifiers declared as variables are assigned an ascending sequence of addresses taken from the interval specified in the working space declaration (see ref. 51). Label identifiers are first assigned zero addresses which are suitably changed when a labelled word is being translated. This mechanism is convenient in checking the proper use of identifiers: at the same level of nomenclature no variable name may be used to label a word, and no label may occur twice; those two conditions are simply checked by examining the address assigned to the name in question.

Ref. 47. The only difference between a local declaration and a declaration is that the former is terminated by a **BEGIN:** on a line of its own, and the latter by a block end. The purpose of a local declaration is to define a level of nomenclature valid only in the block where the declaration occurs. Each level of nomenclature is assigned a new set of locations to avoid errors when a subroutine calls another one, and this still another one and so on. Thus the local declaration do not serve for savings of storage space as in ALGOL. Their practical purpose is to enable the programmer to write a block (a subroutine, say) independently of the identifiers used in the rest of the programme. In practical programmes, however, one or two labels are left nonlocal to ensure easy symbolic communication with the other blocks. This can be also achieved by another method mentioned in ref. 52.

When a clash of a local and nonlocal identifier occurs, the offending nonlocal identifier becomes unavailable.

Ref. 48, 49. A block is a group of lines to be placed in consecutive locations of the store. This group may, or may not, be preceded by local declarations and is terminated with an end of block, which is a right bracket on a line of its own or an asterisk followed by a *lf* (or *cr lf*). The right bracket causes the translator to wait in a keyboard loop, while the other end of block causes the next block to be read without delay. A block end also causes the local forward references to be completed. We call a forward reference every situation when a label identifier occurs in a symbolic address in a line earlier than the one labelled with this identifier.

Ref. 51. The working space declaration serves for two main purposes. First, it specifies the auxiliary locations the translator is allowed to use for its own working purposes, e.g. for name lists and for information necessary to cope with forward references. Second, these locations are to be used by the translated programme to store the values of variables. No block of the programme can be put into this area. The numbers M and N occurring in a working space declaration of the form.

WORKSPACE: $M-N$

are subjected to the condition

$$4 < M < N < 7285-B$$

where B is the number of blocks in the programme being translated. it should be emphasised that the above condition does not prevent the use of the translator's area (locations 7285 onwards) in the translated programme. See also ref. 52.

Ref. 52. The list of block addresses specifies the locations in the store of the beginnings of the consecutive blocks. Each element of the list may be labelled with an identifier. This has the effect that the identifier is assigned the address following it. Thus such an identifier must not be declared at the nonlocal level of nomenclature neither as a label nor as a variable since it is already declared in an obvious sense.

If an address greater than 7284 e.g.

+7500

is found in the list of block addresses, no error is indicated although the location 7500 contains a part of the translator. An alarm message is printed only after a trial to overwrite the location 7500 under the control of the translator. Thus there are easy means to overwrite the translator at run time of the translated programme. Another method of overwriting the translator becomes evident after inspecting the method of address assignment (see example in 4.1).

Ref. 53. The directory of a programme consists of a ∂ sign on a line of its own, the working space declaration, the list of block addresses and declarations (assumed to be nonlocal) and a terminating symbol identical with a block end. The ∂ sign causes some initializations to be done, e.g. releasing the protection of the previously read programme, setting the scaling factor to 1 and so on. If the working space declaration is omitted, the programme is assumed to be written in pure TI code, and thus the syntactic units specific to the AS language must not be used except of titles.

After reading the directory in, the list of block addresses is transformed into the list of allowable store areas for blocks, and the nonlocal identifier list is suitably sorted to enable dichotomic search when reading the rest of the programme. Local identifier lists are sorted when a **BEGIN:** is encountered. A fast sift method is employed, and even in a fairly big programme this process takes only a few seconds.

Ref. 54. An **END:** terminating the symbolic part of the programme, or the whole programme, causes the translator to complete all words so far left incomplete due to nonlocal forward references. After completing the programme the translator comes to a keyboard loop. Further blocks can be read but these must be written in pure TI code. This is of certain value if the translated programme employs library subroutines such as number input and output. For obvious reasons it is convenient to read these subroutines at the end.

It should be emphasised that the **END:** should be punched after a block end. The omission of **END:** in a symbolic programme is a serious error which cannot be detected by the translator if the latter had to be flexible enough.

Special characters used in tape editing. Two consecutive minuses punched after a line of the programme form the end-of-tape symbol and cause the translator to wait in a keyboard loop. If a **?** is punched on the programme tape, the translator ignores all characters since the last *lf*, double minus, left bracket in a trigger and the right bracket which is a block end. It will not operate within an alphanumeric group or title, nor within a copied group.

To erase an incorrectly punched alphanumeric group, a sufficient amount of **?**'s should be punched to bring the number of -non-shift characters after Ł up to seven.

A wrongly punched title can be erased by punching a **?** after the *bl* and before the *lf* terminating the relevant wrong line (in the sense of the syntax table).

3. The AS translator.

3.1 Storage requirements. The AS translator, known in the Programme Library at the Wrocław University under the symbol TU102, uses $874+B$ locations, where B is the number of blocks in the translated programme. When used on a machine with 8192 store, the standard version of TU102 occupies the following locations:

$7285-B$, ..., 7283, 7284	for the list of block addresses,
7285—8046	for the translator's body,
8047—8158	for the main working locations.

The translator uses also the locations specified in the working space declaration (see 2, ref. 51). The last 34 locations of the store are left free, as required by the NE 803 software (the programme T22/23).

3.2. Tapes. Two tapes are provided. The standard tape, marked 803 TU102(CB), is an absolute addressed binary tape for input to standard locations. An AS coded tape, marked 803 TU102(AS), is also available.

3.3. Entry points. An AS programme is always read by reader 1. To commence translating, an entry should be made to any half of the location 7936 (1111100000000). Once put into operation, the translator never comes to a stop, unless a trigger intervenes. If a halt is required (due to the right bracket at the block end, a double minus or a programming error), the translator comes to a keyboard loop (see 3.5).

Another entry point, the first half of the location 7549 (1110101111101), is used to re-read a TI or TICRB coded block. A symbolic block cannot be re-read, as there is no mechanism for removing the information generated when translating labels and symbolic addresses.

The translator can also be used as a subroutine to read a single line (see 3.8).

3.4. Time. The time taken to translate an AS programme varies considerably according to the type of data read. The observed average rate of reading is 120 words per minute (see also (i) in the Introduction). It is worth emphasising that the time of translation is not much sensitive to the number of declared identifiers, as the translator performs only dichotomic searches through name lists.

3.5. Manual control. When the translator is working, it is controlled only via the word generator on the machine's keyboard. The buttons of the NE 803 word generator, essential for operating the translator, are represented below by the letters *G*, *C*, *A*, *L*, *S* (standing for *Go on*, *Copy*, *Addresses*, *Lengths* and *Step-by-step*) and the other ones are represented by the letter 0. Thus, looking at the NE 803 control keyboard

at the time of translation of an AS programme, we should have the following picture:

G 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 *C*
A L 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 *S*

Two keyboard loops are used by the translator. The first one, giving a low and uniform loudspeaker sound, is the NORMAL WAIT used to indicate that either double minus or a right bracket (denoting a block end), or the programme **END:** has been read. The other keyboard loop, giving a loud varying sound, is the ALARM WAIT used to indicate that an error has been revealed in the programme. The ALARM WAIT is preceded by a suitable error message giving some useful details on the error (see 3.6). In any case the translator is made to go on by a change of the *G* button. It is recommended to inspect the error message before going on, to make sure that it is worth doing so (see 3.6).

When the *S* button is kept depressed, the translator comes to the NORMAL WAIT each time a line (in the sense of the syntax table) is read and translated. This is intended to assist in inserting or deleting certain lines of the programme.

If the *C* button is kept depressed, the translator copies each line of the programme on punch 1. Unnecessary shifts, spaces, *cr*'s, *lf*'s, blanks and the words erased with a **?** are removed (except in titles) from the programme.

If the *L* button is depressed, then each time a block end is read (other than the first one at the end of the directory) the translator outputs on the teleprinter (or on punch 2, if direct output is not fitted, or on punch 1 if punch 2 is not fitted) a message of the form

BLOCK *k* **SIZE** *n*

where *k* is the block number of the block just read, and *n* is the number of machine locations occupied by that block, not including the identifiers declared under **VARIABLES:**.

If, on reading the programme **END:** or a block end, the *A* button is depressed, a message of the form

BLOCK *k*
NAME1 DA1 OA1
NAME2 DA2 OA2
. . . . . .
NAME*n* **DA***n* **OA***n*

follows the previous message; **NAME 1, NAME2, ..., NAME***n* are the identifiers declared in the k-th block, and **DA***i* is the absolute decimal address assigned to the i-th identifier. **OA***i* is the octal version of the same address. If $n = 0$, only the message

BLOCK k

is output. This indicates that no identifier has been declared in the k-th block. The list of nonlocal identifiers is printed in the same manner with $k = 0$ on reading the programme **END:**.

3.6. Error indications. All the error messages are printed on the same output device as that used for name lists and block sizes. An error message starts with

ERROR NO. m

where m is the number assigned to the error. This is followed by the last line read by the translator; if $m = 17$ (see the error table), a label identifier is also printed. After the error message is completed the translator comes to the ALARM WAIT (see 3.5). The errors detected by the translator are divided into three types, named A, B, C. All errors of type A or C can be corrected without retranslating the programme from the beginning. To correct such an error, a rectified version of the offending line has to be punched, followed by a double minus, and to be translated as a next part of the programme. If the double minus is omitted, the translator should be set to step-by-step operation (*S* button depressed). The errors of type A and C may also be ignored to obtain the list of errors.

The errors of type B must not be ignored, as they usually cause some important information on the programme to become ambiguous, destroyed or falsified. All errors of type B should be corrected by appropriate change of the source programme and retranslating the whole from the beginning.

The error table

NO.	ERROR
Type A	
1.	A syntactic error in a word, not caused by the use of symbolic addresses.
2.	A number punched wrongly.
3.	An alphanumeric or octal group punched wrongly.
4.	A title violates the title syntax.
5.	An identifier spelt wrongly (or undeclared).
6.	A wrong symbolic address.
7.	More than 70 characters in one line (except in a title or a copied group).

Type B

10. Improper use of storage space (e.g. a trial to overwrite the translator).
11. Too little working space.
12. Working space not declared in a programme with symbolic addresses.
13. A standard word (i.e. **VARIABLES: LABELS: WORKSPACE: END: BEGIN:**) spelt wrongly or an **END:** within a block with local declarations.
14. An identifier declared twice at the same level of nomenclature or the starting character of an identifier is not a letter.
15. Missing **BEGIN:**.
16. Two or more words are labelled with the same identifier at the same level of nomenclature.
17. No possibility of completing a forward reference due to the lack of a labelled word.

Type C

20. All other errors detectable by TU102. They are not listed explicitly as such a list would ever be incomplete. The message printed by the translator is always sufficient to identify the error.

Error no. 11 requires some discussion. If the assignment of addresses to variables requires n locations and m locations are required for the internal purposes of the translator, then $\max(m, n)$ is the minimum number of working locations which should be declared. In most of the real programmes the number n is much greater than m and thus no trouble arises. There may be, however, some long programmes with only a few variables declared. In this case the knowledge of the number m may be of value. It can be calculated as follows: let K be the number of identifiers declared at the nonlocal level and K_i the number of identifiers declared in the i-th block. Further, let F be the number of nonlocal forward references and F_i the number of forward references local in the i-th block. Then the number m is equal to

$$2K + F + \max_i (2K_i + F_i).$$

This is because an element of a name list consists of two locations; the first holds the name itself and the second holds the address assigned to the name. Forward references are handled simply: the relevant instruction is left incomplete, and an appropriate information is stored in working space. This is one machine word containing the following details

(1) which part of the word is left incomplete,
(2) what is the address of the incomplete word,
(3) where the address, not yet known, will be found.

While a block end is being translated, this information is used to complete the local forward references. An **END:** causes the same to be

done with the nonlocal ones. When a label is missing, this is easily detected by inspecting the defined address and **ERROR NO. 17** is displayed together with the offending label name. Once processed, the local information is deleted and the same storage space is used again for the next block. It follows from experience with some practical big programmes, that on a machine with 8192 store even a programme as big as the Elliott Algol translator could be translated, if it were written in the AS language.

3.7. Modifying the translated programme. A temporary directory can be used to modify the translated programme. Although some symbolic information may be contained on the modification tape, the conditions ensuring the correct effect are so severe that, for the sake of safety, relative and absolute addresses should be used throughout. Titles can be corrected and replaced with no restriction. No difficulty arises provided that name lists have been printed (button *A* of the word generator depressed at translation time). Minor corrections can also be done via the control keyboard; octal addresses contained on the name lists are then of great value.

It follows from what was stated previously that a temporary directory, when used within a symbolic programme, may cause strange effects if there is a connection of it with the forward references. To avoid errors it is recommended to use temporary directories after the symbolic part of the programme is completed.

3.8. The AS translator used as a subroutine. The translator can be used as a subroutine in any other programme to read any of the following

(a) an instruction pair, with no symbolic address,

(b) an octal group,

(c) an alphanumeric group,

(d) an integer constant,

(e) a fixed point fraction which will be scaled (in the sense of the TI code) by the contents of the location 8028. The scaling factor is set initially to 1, but it may be changed by any method available to the operator or the programmer,

(f) a floating point number,

(g) a trigger which will be obeyed.

To read one of the above, the programme should contain the instruction pair

```
73 8068 : 40 7289
```

On exit the word just read is found in the accumulator. All the data read are copied on punch 1 if *C* is depressed on the keyboard (see 3.5). All copied groups encountered are also copied.

The translator can also be used to read a title. To do so, the address of the first location to be occupied by the title should be placed in location 7286, and location 8065 should contain the address of the first location which cannot be overwritten. The title punched on the input tape should conform the title syntax and a word of type (a)-(g) above should be punched at the end. On exit the location 7286 contains the address of the last character of the title plus one, and the accumulator contains the final word just read. When the title is too long to be placed in the specified locations, then **ERROR NO. 10** is displayed.

When used as a subroutine, the translator ignores words of the form

$^0/_0$ *cr lf*

4. Complementary information.

4.1. Address assignment. The method used to assign addresses to identifiers is best illustrated by the example below. Dots represent the irrelevant portions of the programme.

```
∂
WORKSPACE: 244—1000
          +5
MAIN) +50
TU12) +100
T105) +132
VARIABLES: I, J, STACK(100), F1(100), F2(20)
LABELS: START, PART
)
VARIABLES: L, L1, I STACK1(100), FF1(10), F2(20)
BEGIN:
START) . . . . . .
. . . . . .
)
VARIABLES: L, M, STACK2(100), X, Y
BEGIN:
PART) . . . . . .
. . . . . .
)
END: lf
```

If a programme with such declarations is translated with the *A* button depressed (see 3.5), the following will be output:

```
BLOCK 1
      I  471  00727
      L  469  00725
     F2  584  01110
     L1  470  00726
    FF1  573  01075
 STACK1  472  00730
BLOCK 2
      L  605  01135
      M  606  01136
      X  708  01304
      Y  709  01305
 STACK2  607  01137
BLOCK 0
      I  244  00364
      J  245  00365
     F1  347  00533
     F2  448  00700
 MAIN     50  00062
 PART     50  00062
 T105    132  00204
 TU12    100  00144
 STACK   246  00366
 START     5  00005
```

4.2. The TU12 subroutine. This is a short (32 words) TI coded subroutine to print titles read and stored by the AS translator. Two entry points are used. We describe them by symbolic exemplification.

Entry 1.

```
73 TU12 : 40 TU12+1
$THIS IS A TITLE TO BE STORED IN CONSECUTIVE LOCATIONS
bl lf
```

If entry 1 is used, the title following the entry instructions is output on punch 1, and the link is increased by an appropriate number to pass the control to the nearest location after the title.

Entry 2.

```
73 TU12 : 40 TU12+2
        : 00 TITADD
```

It is assumed that somewhere in the programme there is a piece of the form

TITADD)$THIS IS A TITLE TO BE STORED IN CONSECUTIVE LOCATIONS *bl lf*

Thus on entry 2 the address of the title is assumed to follow the entry instructions. Having output the title on punch 1 the control is passed to the location link+2.

The first entry is intended to be used when a title is printed at one point of the programme, while the second is of value when the same title is to be output at many points of the programme.

The speed of TU12 on an 803 machine is not less than 96 characters per second. A version of TU12, marked TU12(B), also exists and has four entries to cope with the double Paper Tape Station. TU12(B) is a little longer than TU12.

4.3. Making an absolute addressed binary tape. The 803 software contains an item (T22/23) to cope with the task mentioned above. It is, however, not very convenient to use, especially with the AS translator. We have implemented a modification of T22/23, marked T22/23(KCRB), which is a keyboard controlled relocatable binary tape. This tape can be used independently of any other programme and is fully compatible with T22/23. To input it to locations N onwards, and a part of it to the last 33 locations of the store

40 0 : 00 N

is to be set on the word generator and the machine operated. Then the entry 40 N is used to produce in binary the contents of the specified store locations, and entry 44 N is for checking the tape just produced. All other operating details are just as those for T22/23. The modified tape is by one word longer than T22/23; this is the first word and it contains the instruction pair 40 $N+77$: 40 $N+13$ for reasons obvious to any experienced operator or programmer of the NE 803.

5. Acknowledgement.

The concept of the AS language and the methods for its implementation are due to J. Szczepkowicz. These methods were elaborated in detail by Krystyna Sochacz who also wrote the translator.

The authors are indebted to Dr. S. Paszkowski, head of the Numerical Methods Laboratory (Wrocław University), for valuable discussions on the AS language.

References

[1] K. E. Iverson, *A method of syntax specification*, Comm. ACM 7 (1964), pp. 588-589.

[2] 803 Library Programme T2/102: Translation Input Routine, Supplied by Elliott Bros. Ltd., England.

DEPARTMENT of NUMERICAL METHODS
WROCŁAW UNIVERSITY

Received on 25. 4. 1967

Krystyna SOCHACZ i J. SZCZEPKOWICZ (Wrocław)

OPIS JĘZYKA AS DLA MASZYNY CYFROWEJ ELLIOTT 803

STRESZCZENIE

W pracy opisano język adresów symbolicznych AS opracowany i zrealizowany przez autorów dla maszyny Elliott 803 w celu uproszczenia prac związanych z przygotowaniem programów w kodzie wewnętrznym tej maszyny. Istotne cechy języka AS są następujące:

1. Używany dotąd kod TI maszyny Elliott 803 jest podzbiorem języka AS.
2. Słowo rozkazowe języka AS jest tłumaczone na jedno słowo rozkazowe maszyny.
3. W języku AS można używać nazw rozumianych tak, jak w ALGOLu.
4. Nazwy w programie mogą być opisane jako zmienne lub etykiety. Etykietę można umieścić przed każdym słowem rozkazowym, przed każdą stałą i tytułem.
5. Obszarem działania nazwy może być cały program albo tylko pewien jego fragment zwany blokiem. Umożliwia to np. korzystanie z podprogramu (napisanego w języku AS) bez żadnych ograniczeń dla nazw używanych w pozostałej części programu.

Praca zawiera opis składni języka AS w postaci formuł składniowych Backusa-Naura, skróconych metodą Iversona, oraz istotne szczegóły dotyczące translatora dla tego języka.

Кристина СОХАЧ и Й. ЩЕПКОВИЧ (Вроцлав)

ОПИСАНИЕ ЯЗЫКА AS ДЛЯ ВЫЧИСЛИТЕЛЬНОЙ МАШИНЫ NE 803

РЕЗЮМЕ

В работе описывается язык символических адресов AS, разработанный авторами для облегчения подготовки программ в машинном коде вычислительной машины NE 803. Основные свойства этого языка следующие:

1. Код TI машины NE 803, который употреблялся до сих пор, является подмножеством языка AS.

2. Любой команде языка AS соответствует одна машинная команда.

3. В языке AS используются идентификаторы определённые так как в АЛГОЛе.

4. Идентификаторы могут быть описаны как переменные или метки. Всякая команда, константа или текст может иметь метку.

5. Любой идентификатор может быть глобальным в программе или локализованным в некоторой части программы — в блоке. Поэтому подпрограмма может быть использована в произвольной программе без изменений идентификаторов с этой программы.

Работа содержит синтаксическое описание языка AS методом Бэкуса-Наура, сокращённым методом Иверсона, и некоторые важыне свойства транслятора с этого языка.