EUR 5173 e

COMMISSION OF THE EUROPEAN COMMUNITIES

THE SLC-II LANGUAGE TRANSLATION PACKAGE USER MANUAL FOR INPUT

by

W. KOLAR

1974

Joint Nuclear Research Centre
Ispra Establishment - Italy

This report is on sale at the addresses listed on cover page 4

at the price of B.Fr. 125,—

Commission of the
European Communities
D.G. XIII - C.I.D.
29, rue Aldringen
Luxembourg

September 1974

EUR 5173 e

THE SLC-II LANGUAGE TRANSLATION PACKAGE
USER MANUAL FOR INPUT by W. KOLAR

Commission of the European Communities
Joint Nuclear Research Centre - Ispra Establishment (Italy)
Luxembourg, September 1974 - 98 Pages - 1 Figure - B.Fr. 125,—

The present publication is the user manual of the input program of the SLC-II system. It gives the functional description of the program and describes the different instructions which are at the disposal of a user to code his problem. An example illustrates the use of the program.

EUR 5173 e

THE SLC-II LANGUAGE TRANSLATION PACKAGE
USER MANUAL FOR INPUT by W. KOLAR

Commission of the European Communities
Joint Nuclear Research Centre - Ispra Establishment (Italy)
Luxembourg, September 1974 - 98 Pages - 1 Figure - B.Fr. 125,—

The present publication is the user manual of the input program of the SLC-II system. It gives the functional description of the program and describes the different instructions which are at the disposal of a user to code his problem. An example illustrates the use of the program.

EUR 5173 e

THE SLC-II LANGUAGE TRANSLATION PACKAGE
USER MANUAL FOR INPUT by W. KOLAR

Commission of the European Communities
Joint Nuclear Research Centre - Ispra Establishment (Italy)
Luxembourg, September 1974 - 98 Pages - 1 Figure - B.Fr. 125,—

The present publication is the user manual of the input program of the SLC-II system. It gives the functional description of the program and describes the different instructions which are at the disposal of a user to code his problem. An example illustrates the use of the program.

EUR 5173 e

COMMISSION OF THE EUROPEAN COMMUNITIES

THE SLC-II LANGUAGE TRANSLATION PACKAGE
USER MANUAL FOR INPUT

by

W. KOLAR

1974

Joint Nuclear Research Centre
Ispra Establishment - Italy

ABSTRACT

The present publication is the user manual of the input program of the SLC-II system. It gives the functional description of the program and describes the different instructions which are at the disposal of a user to code his problem. An example illustrates the use of the program.

CONTENTS

THE SLC-II LANGUAGE TRANSLATION PACKAGE
USER MANUAL FOR INPUT

W. Kolar

Remark: It is assumed that the user of this manual is familiar with the publication: "Concepts and Facilities"[(1)]

1. INTRODUCTION

TEXTANAL is the input module of the SLC-II system. Its function is:

1. to read the input text,
2. to extract from the input text the word- and non-word items,
3. to eliminate all insignificant characters or character sequences from further processing,
4. to list optionally the input text and the word items.

The basic concept of TEXTANAL is to consider the input text as a sequence of structured text units, called henceforth "objects" (Fig. 1). The different elements of such an object shall be called "components". Components may be divided into smaller components. A component, which is not further divided, is a "terminal component".

Some examples may clarify this principle. An object may be a book, a bibliographic unit etc. The components of a book may be the chapters, which themselves are subdivided into paragraphs. These paragraphs may be considered to be terminal components. One can choose, of course, other segmentations, which will lead to different components. A bibliographic unit can consist of the author, the title, the abstract, the descriptor component etc. Some of these components may be again subdivided e. g. the author component could consist of a name and an affiliation subcomponent, the title component could be made up out of the English title and a title in the original language.

The analysis of an object is performed on two levels, the "format level" and the "data level".

1. The format level describes the structure of an object by means of the "format grammar". There may exist only one format grammar per problem case.
2. On the data level the contents of a terminal component are analyzed by the "data grammar". Different terminal components may be analyzed with different "data grammars".

The grammars have to be coded by the user. They consist of:

1. the format syntax table,
2. one or more scanner tables,
3. one or more data syntax tables,
4. the semantic action table.

The format syntax table contains the information of the SLC-instructions which are used to describe the syntax of the formal level. (Chapter 2.1). Each instruction corresponds to one table entry.
The scanner tables contain the characters of the source text alphabet, to which a delimiter function for the scanning process has been attributed.
The data syntax tables contain the information of the SLC-instructions describing the syntax of terminal components (Chapter 2.2). Each instruction corresponds to one table entry.

The semantic action table contains the SLC-instructions, which are at the user's disposal to control and/or to manipulate the input text character or character sequences (atoms) which have been accepted by the syntax. Each instruction corresponds to one table entry.

At execution time these tables are linked to interpretive programs, which are:

1. a scanner program,
2. two different parser programs;
   a "format parser" to analyze the "format syntax",
   a "data parser" to analyze a "data syntax";
3. a semantic interpreter.

The contents of the table entries are analyzed and the program flow is directed according to the coded information. Only one table may be linked to each processor at one time.

The tables of the format parser and the semantic interpreter are linked without user intervention, while the tables of the scanner and the data syntax have to be specified in an instruction of the format syntax (ANALYSE).

2. FUNCTIONAL DESCRIPTION

2.1 The Format Level

The format grammar comprises the format parser, the semantic interpreter and the corresponding tables.
Its function is:

1. to read the input text,
2. to identify the terminal components,
3. to select the scanner and data syntax tables and link them to the processer programs,
4. to branch to the data level.

Seven SLC-instructions INPUT, OUTPUT, PARSE, GETADR, VAL, ANALYSE and PRØGEND may be used to describe the syntax of this level.

INPUT reads a logical record from an external device into the memory. OUTPUT prints the contents of a string (strings are generated by the instruction GENSTR, see chapter 2.3). PARSE is a conditional or unconditional branch instruction within the format syntax tree. It is the only instruction on the format level, which allows the execution of semantic action routines. GETADR calculates the memory addresses of terminal components if the corresponding displacements with respect to a start address are known. The instruction VAL allows the manipulation (addition) of numerical values. This instruction may be used for address calculations. ANALYSE links the chosen scanner and data syntax tables to the interpretive programs, specifies the input text field to be analyzed, performs if necessary the concatenation of the rest of a record with the successive one and executes the branch to the data level. PRØGEND is an instruction which stops the analysis of the input text and terminates the program execution.

The format grammar may be considered as a kind of supervisor of the program TEXTANAL. Execution of TEXTANAL starts always on the format level. From here the program branches to the data level and returns to the instruction of the format syntax, which follows the instruction ANALYSE.

2.2 The Data Level

The data grammar consists of the scanner program, the data parser, the semantic interpreter and the corresponding tables. The function of this level is to generate strings, which are either:

1. word-items,
2. non-word items,
3. or character strings to be ignored in the further processing.

Five SLC-instructions INSCAN, MULTSCAN, TERM, NOTERM and BRUNC are defined for this level.

INSCAN and MULTSCAN are the instructions which allow one to attribute to characters of the source text alphabet a special meaning in the scanning process. One can distinguish three functional classes of characters:

1. Characters, which are syntactic elements for the parser (individual atoms). The characters are defined with the instruction INSCAN.
2. Characters, which, if there is a consecutive sequence of them, are combined into <u>one</u> syntactic element (multiple atom). These characters are defined with the instruction MULTSCAN.
3. All the characters, which are not defined by an instruction INSCAN or MULTSCAN.

The input text is considered to be an infinite string of characters. The task of the scanner is now to construct those substrings which are syntactic elements (atoms) for the parser. Characters of the first and second class are by definition syntactic elements. They interrupt the scanning process when they are encountered. Characters of the third class are accumulated until a character of one of the other types is detected. Then the scanning process is interrupted and the character sequence is transferred as syntactic element (word atom) to the parser.

TERM, NOTERM and BRUNC are used to describe the data syntaxes. TERM defines the possible syntactic elements of a node of the syntax tree. For each syntactic element one instruction has to be coded. NOTERM has a function similar to that of a CALL instruction of a high level language. It forces the branch to a subtree of the syntax tree with subsequent return to the original point. BRUNC is an unconditional branch instruction.

The parsing process may be described in the following way. When an atom is passed from the scanner, the parser is activated. It interprets the table entry of the syntax table to which it points. If the entry is an instruction TERM, the scanned atom is compared with the atom defined by the instruction (parameter field COD=). If the two atoms are matched, a semantic action routine (parameter field ACT =) is executed if present and the program then branches to the node of the syntax, which is specified in the parameter field NEXT = . If the atoms are not matched, the parser proceeds to the next sequential instruction (next table entry).

If the table entry is an instruction NOTERM, the program branches to a subtree. Nesting of NOTERM instructions is allowed (≤ 20). A semantic action, coded in a NOTERM instruction is only executed on return from the subtree.

2.3 Generating and Auxiliary Instructions

The reservation of memory areas is performed with the instruction GENSTR. The instruction allows the creation of named strings from 1 byte to 32,767 bytes length. GENTAB is the instruction which is used to create tables. At execution time these tables serve for translation and control purposes. Translation means in this context, the replacement of a string by another one. FORM is the instruction used to describe the components of an object. SYNTAX denotes the beginning of a syntax table. ERMESS is an instruction, which creates a table of error messages. This table is used together with the instruction ERROR of the semantic interpreter to list error conditions.

3. RESULTS

The output of TEXTANAL may consist of:

1. a list of the processed input text,

2. a list of the different word items encountered during analysis,
3. five data files with the DDNAMES:

   a) TEXTABLE
   b) WORDTAB
   c) WORDFILE
   d) FREQFILE
   e) FREQTAB.

The printing of the input text (instructions OUTPUT or PRISTRI may be suppressed, if a flag bit on an input card of SLCMONIT (monitor program of the SLC-II system) is set to zero. In the same way one can decide on the output of the word items.

The information stored in the different data files reflect the internal program organization of TEXTANAL. TEXTANAL contains two tables WORDTAB and WORDS (see example in "Concepts and Facilities"). WORDTAB has an entry for each different word item. This entry has two pointers, one to the table WORDS and one to the next word item. WORDS contains the word items. They are stored in the order of their first occurrence in the input text. Each word item is preceded by its length indicator (number of characters -1).

The data file TEXTABLE contains an image of the input text. Word items are represented by pointers to the table WORDTAB (entry number), non-word items are stored in their original form together with a flag and their length indicator. Word items and non-word items are aligned to halfword boundaries. The data files TEXTABLE, WORDTAB and WORDFILE allow the reconstruction of the input text; e.g. after the translation of the word items from the source language into the target language.

The data files FREQFILE and FREQTAB are optional. If they are to be used, a flag bit on the input card OPTIONS of the program SLCMONIT, has to be set to one. On FREQFILE are stored the word items together with their frequencies. This data set may be used in off-line SORT and MERGE applications. FREQTAB saves the table of frequencies of the word items ordered as in the table WORDTAB.

Appendix No. 2 contains a list of the different DD-statements. If one data set is not needed, the DD-statement may be replaced by a dummy DD-statement. The optional data sets may be omitted. (Note, the option bit must be set zero).

4. PROGRAMMING CONSIDERATIONS

In coding the different grammars (the program), the user must respect two rules:

1. The instructions GENSTR have to be the first statements of the program.
2. The a) semantic action routines (instr. CATEN, SUB, SEND, etc.),

   b) data syntax tables (instr. TERM, NOTERM, BRUNC)

   c) format syntax (instr. INPUT, OUTPUT, PARSE, etc.)

   must be placed in the order defined above at the end of the program.

A possible arrangement of the different program parts may be:

1. the instructions GENSTR,
2. the tables GENTAB,
3. the scanner tables (INSCAN, MULTSCAN),
4. the instruction ERMESS,
5. the instruction FORM,
6. the action routines,

7. the data syntax tables,
8. the format syntax.

Appendix No. 5 shows an example of a user-program and the results (input text, list of word items).
The user-program is assembled with the assembler H, link-edited and stored as an executable load module member in a partitioned data set.

Execution of TEXTANAL is initiated by a call from the supervisor program SLCMONIT. TEXTANAL then calls the user-program into memory. It is possible to combine different user-programs with TEXTANAL. For this the member name of the load module has to be coded in one of the monitor input cards. Appendix No. 2 lists the control statements, needed to create the load module and to execute TEXTANAL.

At assembly time two types of diagnostics are issued in the case where an error is detected:

1. The error messages generated by the assembler H (IBM brochure NR. SC6-3770-1)
2. The error diagnostics of TEXTANAL. Appendix No. 3 explains each message in detail.

It may happen that both types of messages are printed for the same error.

At execution time erroneous program situations may arise due to wrong input data or incomplete grammars. In order to facilitate the location of these errors within the program, error diagnostics are printed before the program terminates abnormally. Appendix No. 4

explains the error messages.

If the amount of source text is too large to be processed in one cycle, the input analysis is interrupted, the analyzed data is processed by the subsequent modules, and TEXTANAL continues execution with the next batch of data. Optionally the analysis cycle may be terminated after one text batch. A restart facility allows one to continue the analysis in a later run. For this the number of records to be skipped and the displacement in the first record where analysis restarts have to be specified on a monitor input card. Both values are listed on the output list of the preceding run. One should note that the displacement field may only be specified when <u>one</u> input text field has to be analyzed. In the case where different input text fields (terminal components) exist, the program can only restart with a complete record. Appendix No. 2 shows also the control statements for this application.

ACKNOWLEDGEMENT

The author thanks Mr. PERSCHKE for the discussions and suggestions especially at the beginning of this work. He is very much indebted to Mr. FANGMEYER for numerous clarifying discussions and the first critical evaluation of the results.

It is to be mentioned that a first version of TEXTANAL has been prepared under contract by the Software firm ITALSIEL.

REFERENCES

(1) "The SLC-II Language Translation Package: Concepts and Facilities". EUR-Report to be published.

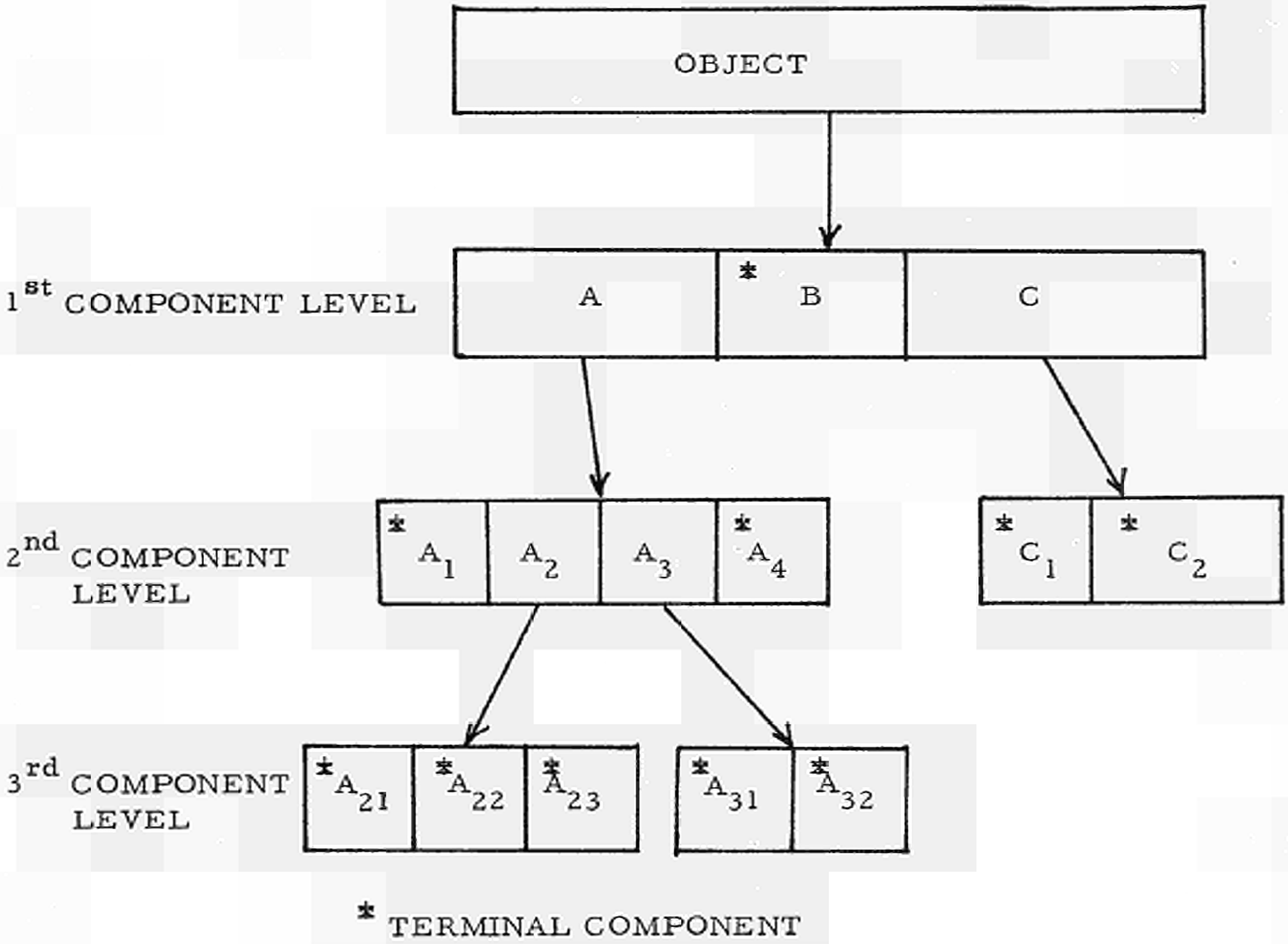

FIG. 1 - BLOCK DIAGRAM OF A STRUCTURED OBJECT

APPENDIX No. 1

The appendix describes the SLC-instructions which are at the disposal of the user.

THE SLC INSTRUCTIONS OF TEXTANAL

1. Coding Conventions and Abbreviations

The SLC instructions are coded according the rules of assembler language coding (IBM Manual GC 28-6514). Following these conventions a SLC instruction consists of four fields: a name field, an operation field, an operand field and a comment field. The fields must be separated by at least one blank.

The name field contains a symbol or is blank. A symbol corresponds to the "ordinary symbol" of assembler language coding, what means:

1. the first character must be a letter (A-Z, $, #, @);
2. no special characters or blanks are allowed;
3. digits are allowed.

If a symbol is coded in the name field of an instruction, it is called henceforth "label".
The operation field contains the mnemonic operation code. It must be written as shown.
The operand field contains the operands from which a subset may be selected according to the desired action. In order to describe the different types of operands the following notation has been adopted:

1. The brackets [ ] indicate that the operand is optional;
2. The braces { } indicate that a choice out of the stack has to be made;
3. The sequence ,.... indicates that the operand may be repeated.

The operands have to be coded according to the following rules:

1. Operands written in upper case letters must be coded as shown;
2. Operands written in lower case letters should be replaced by the

specific information;

3. Operands which are a combination of both types separated by an equal sign have to be coded in the following way (keyword operands):

   3.1 Capital letters and equal sign exactly as shown;

   3.2 Lower case letters have to be substituted;

   3.3 Digits have to be coded as shown;

4. Commas, parenthesis or quotes have to be coded as shown. (The comma after the last operand should be omitted).

The <u>comment field</u> may contain any combination of characters.

In the discussion of the instructions of TEXTANAL the following abbreviations will be used:

- lab	label of an instruction,
- sym	symbol,
- sr, srl, ...	names of stringes,
- val, vall, ...	integer values,
- sem, seml, ...	label of semantic instruction,
- nod, nodl, ...	label of a syntax instruction,
- const, constl, ...	character, hexadecimal or binary constant (*),
- tab	name of a table,
- op	one of the following relational operators GT, GE, EQ, LE, LT, NE.

(*) The maximum length of a constant is 256 bytes. A character constant consists of the letter C followed by the characters enclosed in quotes (e.g. C'$#CAT#$'). A hexadecimal constant consists of the letter X followed by hexadecimal digits enclosed in quotes (e.g. X'123BA4'). A binary constant consists of the letter B followed by a sequence of 1's or 0's enclosed in quotes (e.g. B'10011100').

Note: An ampersand and a quote used in a character constant must be coded as two ampersands and two quotes respectively, (e.g. C'A&&' or C'C''').

2. The SLC-Instructions of the Format Level

INPUT:

The format of the instruction is:

name	operation	operand
lab or blank	INPUT	ADR = sr

INPUT reads a logical record from an input device into the core and stores the address of the buffer in the string sr.
Example:

```
INPUT        ADR = ADBUF
```

A record is read and the buffer address is stored in the string ADBUF.

OUTPUT:

The format of the instruction is:

name	operation	operand
lab or blank	OUTPUT	STRING = sr

OUTPUT prints the contents of a string. Sr is the name of the string to be listed. If the string is an empty string, the message ***WA1*** (see Appendix No. 4) is printed and execution <u>continues</u>.
If the length of the string exceeds 132 characters, the message ***ER11*** is printed and the execution of TEXTANAL is <u>terminated.</u>

Example:

1. OUTPUT STRING = INPUT

The input record is printed. The stringname INPUT is defined by the system at generation time. It need not be generated by an instruction GENSTR. The user may use this instruction to print input data.

2. OUTPUT STRING = STR1

The contents of the string with name STR1 is listed. It should be kept in mind that the character of the string should be in printable format.

PARSE:

The format of the instruction is:

name	operation	operand
lab or blank	PARSE	$\left[\left\{\begin{matrix} \left\{\begin{matrix} \text{val} \\ \text{sr} \end{matrix}\right\} \text{ op } \left\{\begin{matrix} \text{val} \\ \text{sr} \end{matrix}\right\} \\ \text{sr} \left\{\begin{matrix} \text{EQ} \\ \text{NE} \end{matrix}\right\} \left\{\begin{matrix} \text{ON} \\ \text{OFF} \end{matrix}\right\} \end{matrix}\right\},\right]$ [ACT=sem,] [NEXT = nod,]

PARSE may be used as conditional or unconditional branch instruction within the format syntax. No branch from the format syntax to a data syntax is allowed.

If PARSE is used as a conditional branch instruction, the positional operand (first bracket) has to be present. It represents a relational expression with the values "true" or "false". If the value is "true", the action routine sem (see chapter 5 of this Appendix) is executed, if present, and the parser branches to the syntax instruction labelled nod. If the value is "false", the parser proceeds to the next sequential instruction of the syntax.

The allowed relational operators (op) are: GT, GE, EQ, LE, LT, NE; val may be an integer value: $-32768 \leq \text{val} \leq 32767$.

There are two different applications of the positional keyword parameter:

1. Comparison of a string with a numerical value or with another string.

2. Test of the status of a switch.

If the string sr is compared with the value val, sr must be a word-string (4 bytes) (see instruction GENSTR). If the string sr is compared with another string sr, both strings must have equal length, otherwise the message ***ER13*** is printed and the job is terminated. Comparison of the two strings is done character by character. Any character is allowed.

To test the status of a switch only the relational operators EQ and NE may be used. ON or OFF are the possible states of the switch sr, which must be defined as a word-string (4 bytes).

Examples:
1. PARSE 'COUNT LT 2048', ACT=AR1, NEXT=N12
2. PARSE 'STR1 EQ STR2' , ACT=AR2
3. PARSE 'SWI1 NE ON' , NEXT=N13
4. PARSE ACT=AR2, NEXT=N12
5. PARSE NEXT=N12
6. PARSE ACT=AR3

The first three examples show the use of the instruction PARSE as a conditional branch instruction, the last three as an unconditional one.

In the first example PARSE checks if the numerical value stored in the string COUNT is smaller than 2048. Is this the case, the semantic action AR1 is executed and the parser branches to syntax instruction labelled N12, otherwise the parser proceeds to the next sequential instruction.

In the second example, the contents of the two strings STR1 and STR2 are compared. If they are equal, the action AR2 is executed.

The third example controls the status of the switch SWI1. If the switch is in the OFF-status, the parser branches to the point N12 of the syntax.
The functions of the examples of the unconditional use of PARSE correspond to those of the conditional use.

GETADR:

The format of the instruction is:

name	operation	operand
lab or blank	GETADR	ADR=sr, $\left\{\begin{array}{l}\text{COMP=sym}\\\text{FIELD=sym}\end{array}\right\}$

GETADR is an instruction which permits:

1. the calculation of the memory addresses of terminal components of an object;
2. the assignment of an address to a terminal component.

The terminal components are defined by the instruction FORM.
Sym is the label of an instruction FORM, sr the name of a string which contains an address.
If one chooses the parameter COMP=sym, GETADR determines the memory addresses of all those components, which are coded between the instruction FORM labelled sym and the next labelled instruction FORM (the second labelled instruction is excluded). The displacements of all the components, in respect to the address (in string sr) have to be known.

The second version is executed if the parameter FIELD = sym is coded. sr defines the string, containing the address of the field, sym is the name of a string or component created with the instructions GENSTR and FORM respectively.

Examples: 1. GETADR ADR = ADBUF , COMP = F1

2. GETADR ADR = ADSTR , FIELD = TITLE

The description of a component may be the following:

```
F1    FORM    (IDEN, 8, 0)
      FORM    (CAT, 1, 0)
      FORM    (REF, 5, 6)
      FORM    (DATA, 76, 12)
F2    FORM    (LENGTH, 12, 0)
F3    FORM    (TITLE, 27)
```

In the first example GETADR calculates the addresses of the components IDEN, CAT, REF and DATA, assuming that the string ADBUF contains the starting address.
In the second example the address contained in string ADSTR is assigned to the component or string named TITLE.
After the assignments of the actual addresses to the components, they may be considered as strings and used in any instruction permitting strings in its operand field.
(The length of the components must be known and coded in the instruction FORM!).

VAL:

The format of the instruction is:

name	operation	operand
lab or blank	VAL	OLD = sr1, NEW = sr2, INCR = $\left\{ \begin{matrix} \text{val} \\ \text{sr3} \end{matrix} \right\}$

VAL is an instruction which allows arithmetic manipulation (addition) of numerical values. The values are contained in strings. sr1 is the string which contains the value to be updated; sr2 is the string which will contain the result of the operation. The third parameter INCR defines the value to be added to the value given in sr1. INCR may be an integer constant $-32768 \leqslant val \leqslant 32767$ or the name of a string sr3. In this case the maximal integer value may be $2^{31}-1$ (214 748 647).
sr1, sr2 and sr3 may specify the same string.

Examples: 1. VAL OLD = STR1, NEW = STR2, INCR = 25

2. VAL OLD = S1, NEW = S1, INCR = STR4

In the first example the value of STR1 is increased by 25 and the result is stored in STR2.
In the second example the value of the string STR4 is added to the value of string S1 and the result is stored in string S1.

ANALYSE:

The format of the instruction is:

name	operation	operands
lab or blank	ANALYSE	SCAN=sym1, SYNTAX=sym2, TEXT=sym3

ANALYSE links the scanner table named sym1 and the data syntax table named sym2 to the scanner and data parser program, respectively. The third parameter TEXT = sym3 specifies a name of a string or component to be analyzed. If the length of the field is zero, the error message ***ER14*** is printed and the execution is terminated.

ANALYSE performs also the concatenation of two successive records. If the first one terminates with a word-atom, the program transfers it to a special area (500 bytes) and adds the next record. Scanning restarts with the word-atom. If the concatenated record would exceed the length of the reserved area, the error message ***ER12*** is printed and the execution of TEXTANAL is terminated.

It should be noted that concatenation is permitted only in the case where <u>one</u> data field is analyzed. It is the user's responsibility to respect the correct application of this rule.

After such a concatenation, ANALYSE branches to the data level.

<u>Example:</u> ANALYSE SCAN=SCAN1, SYNTAX=SYN1, DATA=DAT1

PROGEND:

The format of the instruction is:

name	operation	operand
lab or blank	PROGEND	

PROGEND terminates the execution of TEXTANAL.

<u>Example:</u> N15 PROGEND

3. <u>The SLC-Instruction of the Data Level</u>

INSCAN, MULTSCAN:

The format of the instructions is:

name	operation	operand
lab or blank	$\left\{ \begin{array}{l} \text{INSCAN} \\ \text{MULTSCAN} \end{array} \right\}$	(sym, const),...

INSCAN defines the characters of the source text alphabet which are syntactic elements in the parsing process, (individual atom).

MULTSCAN defines the characters which are also syntactic elements with the special feature, that sequences of equal characters are represented by <u>one</u> element (different characters defined by a MULTSCAN instruction are different elements for the parser).
Lab is a label (symbol) and defines the beginning of a scanner table. A scanner table comprises all the instructions between two labelled instructions INSCAN or MULTSCAN (the second one excluded).

Sym must be a valid symbol. It represents the name attributed to the character defined in const. The name sym is referenced in the parameter field COD=sym of the instruction TERM. Const is a one-character constant in binary-, hexadecimal or character format.

<u>Example:</u>

```
SCAN1   INSCAN    (PLUS, C'+')
        INSCAN    (COMMA, C',')
        MULTSCAN  (BLANK, C' ')
SCAN2   INSCAN    (PERIOD, X'4B')
        INSCAN    (FIVE, C'5')
        MULTSCAN  (BLANK, C' ')
        MULTSCAN  (FEED, X'FF')
```

In this example two scanner tables named SCAN1, SCAN2 are coded. The first table comprises the first three statements and defines the characters + and , as individual atoms and the blank character as multiple atom. The second table (last four instructions) defines the characters X'4B' (period) and the number 5 as individual atoms and the blank character and the character X'FF' as multiple atoms. Note, that a sequence of blank characters followed by a sequence of characters X'FF', are treated as two different multiple atoms.

TERM:

The format of the instruction is:

name	operation	operand
lab or blank	TERM	COD = $\begin{Bmatrix} \text{sym} \\ \text{WORD} \end{Bmatrix}$, NEXT = nod, [ACT = sem,] [ERC = val,] [STOP = 1,]

TERM defines the possible syntactic elements which may be present in a node of the syntax. A node comprises all the instruction TERM, NOTERM and BRUNC coded between two labelled instructions (the second one is excluded).

For each syntactic element one instruction has to be coded. If the syntactic element is a word-atom, one must code COD = WORD in all other cases COD = sym. Sym is the name attributed to an individual atom or multiple atom.

TERM compares the scanned atom with the atom coded in the keyword operand COD. If the two atoms agree, the semantic action as specified in ACT = sem is executed and the parser branches to the label nod of the syntax as given in the keyword operand NEXT =. STOP = 1 may be coded if an instruction TERM is the last instruction of a node. In this case also the parameter ERC = val should be coded ($1 < \text{val} \leqslant 63$). If the parser does not find agreement between the scanned and coded atoms for all the instructions TERM of the node, the last one flags an error condition. The error message ***ER15*** together with the value val is printed and the job is terminated abnormally.

Resumé of the different parameters:

1. sem label of a semantic action routine,

2. nod label of a syntax instruction,

3. sym name attributed to an atom (syntactic element),

4. val integer value 1 $<$ val $\leqslant$ 63.

<u>Example:</u>

```
NOD1 TERM ACT=S1,NEXT=NOD5,COD=WORD
     TERM NEXT=NOD2,COD=COMMA
     TERM NEXT=NOD5, COD=PERIOD, STOP=1,ERC=2
NOD2 TERM NEXT=NOD5,COD=BLANK
          .
          .
          .
```

The node NOD1 comprises the first three instructions. The first instruction executes the semantic action S1 and branches to node NOD5, if the scanned atom is a word atom, otherwise the parser proceeds to the second instruction. If it does not match the scanned atom with one of the coded ones, the program prints an error message ***ER15*** SYNTAX ERROR NR 2' and terminates abnormally. If the scanned atom is matched in the third instruction, the program branches to NOD5.

NOTERM:

The format of the instruction is:

name	operation	operand
sym or blank	NOTERM	nod, nod1 [, ACT = sem]

The instruction NOTERM is a branch and link instruction. nod and nod1 are labels of the syntax, sem is the label of a semantic action routine.

nod1 is the entry point of the subtree to which the parser has to branch and nod is the label from which execution will continue, on return from the subtree. nod and nod1 are two mandatory parameters.

The semantic action sem is only executed on return of the program from the subtree.

Examples: 1. NOTERM NOD5, TREE1

2. NOTERM NOD6, TREE2, ACT = AR5

The first instruction signifies, that the parser branches to a subtree TREE1 and continues execution on return at label NOD5. In the second instruction the action AR5 is executed on return from the subtree TREE2 before execution continues at label NOD6.

BRUNC:

The format of the instruction is:

name	operation	operand
sym or blank	BRUNC	$\left\{\begin{array}{l}\text{NEXT = nod [, ACT = sem]}\\ \text{EXIT = 1 [, ACT = sem]}\\ \text{TYPE = 3 [, ACT = sem]}\end{array}\right\}$

BRUNC is an unconditional branch instruction. nod is a label within the data syntax table and sem a semantic action routine.

The first version of BRUNC is used to describe a branch within the data syntax. No branch into a subtree is allowed. The second version is used to branch back from a subtree into the main tree. It may be used only in subtrees. The third version forces a branch from a subtree to that instruction of the main tree, which follows the NOTERM instruction, originally initiating the branch. It may be used only in subtrees.

Examples: BRUNC NEXT = NOD5, ACT = ROUT

BRUNC NEXT = NODEND

In the first example the program executes the semantic action routine ACT = ROUT and branches to the instruction labelled NOD5. In the second example it branches to NODEND.

In order to explain the second and third version of the instruction, the following syntax is assumed.

MAIN Syntax Tree

```
NODE17   TERM     ACT=AR2,NEXT=NODE18,COD=HYPHEN
         TERM     ACT=AR3,NEXT=NODE18,COD=MBLANK
         TERM     ACT=AR26,NEXT=NODE21,COD=$
         NOTERM   NODE16,CWS,ACT=AR3
RT3      BRUNC    ACT=AR22,NEXT=NODER11

                  SUBTREE

CWS      TERM     ACT=AR22,NEXT=CWS,COD=COMMA
         TERM     ACT=AR2,NEXT=CWS,COD=WORD
         TERM     NEXT=NODE46, COD=MBLANK
         BRUNC    TYPE=3
NODE46   BRUNC    EXIT=1
```

If the scanned atom is neither HYPHEN, MBLANK nor $, the program branches, in executing the 4th instruction, to the subtree named CWS and loops in it as long as the scanned atoms agree with the COD parameters COMMA and WORD. If the first scanned atom, different from COMMA and WORD, is MBLANK, the program branches to NODE46 and returns from here to the main syntax tree (NOTERM instructions). The action AR3 is executed and the program branches to NODE16. If however, the scanned atom is not MBLANK, the program executes the 4th instruction of the subtree (BRUNC TYPE=3) and branches to the instruction labelled RT3 in the main syntax tree, where it continues the execution.

4. The Generating and Auxiliary Instructions

GENSTR:

The format of the instruction is:

name	operation	operand
lab or blank	GENSTR	{ sym, ... (sym, val), ... (sym, const), ... }

GENSTR is the instruction which reserves named contiguous areas of core storage. Sym is the name attributed to the string. A string is referenced by coding its name. The instructions GENSTR must be the first of a user program. Two strings with name WATOM and INPUT are generated automatically.

1. WATOM is the string where the scanner stores the atoms.
   The user may use the string WATOM together with the semantic instructions (see chapter 5 of this Appendix) to manipulate the scanned atom.

2. INPUT is the string which contains a logical input record.

GENSTR allows the creation of three types of strings.

1. Word-strings (4 bytes)
   They are coded

   GENSTR   sym1, sym2, ...

   Sym1, sym2 are symbols. The reserved core locations are cleared.

2. Explicit-length strings
   They are coded

   GENSTR (sym1, val1),(sym2,val2),....

   Sym1, sym2 are symbols, val1, val2 are integers. ($1 \leqslant$ val $\leqslant 32767$).

GENSTR reserves a contiguous area of val bytes in the core. The areas are not cleared.

3. Initialized strings

They are coded

GENSTR (sym1, const1),(sym2,const2),....

Sym1, and sym2 have to be valid symbols. const.1 and const2 represent character, hexadecimal or binary constants.

GENSTR creates a string with the constant as its contents. The length of the string is determined by the type of the constant.

The different kinds of strings may be coded in any order.

Examples: GENSTR ADBUF, SWI1, STRBUF, COUNT1
GENSTR (STR1,256),(REF,C'##RIF##')
GENSTR (WCAT,1),(HYPHEN,X'4X')
GENSTR (BLANKL, CL122' ')

Note: the instruction

GENSTR (BLANKL, 122CL1' ')

is not valid.

GENTAB:

The format of the instruction is:

name	operation	operand
lab or blank	GENTAB	(const1, const2),....

GENTAB is the instruction which creates tables for control and "translation" purposes. A table consists of all the instructions GENTAB, coded between two different labelled instructions (the second labelled instruction is excluded).

Constl and const2 are character, hexadecimal or binary constants. If a table is used for control purposes at execution time (LOOKTAB), the constant constl is searched. If the table is however used for "translation" purposes (TRANS), the contents of the string, which is originally constl, is replaced by the string const2.

<u>Examples:</u>

```
  WIDTAB   GENTAB  (C'M',X'10')
           GENTAB  (C'SL,S',B'00110101')
           GENTAB  (X'6A',C'HELP')
 NWIDTAB   GENTAB  (X'35',C'ALZ,S5')
 NWIDTAB   GENTAB  (B'11101100',C'BS7')
   LDTAB   GENTAB  (X'A',X'25')
```

WIDTAB is defined by the first three instructions, NWIDTAB by the following two and LDTAB by the last instruction.

FORM:

The format of the instruction is:

name	operation	operand
lab or blank	FORM	(sym, vall, val2)

FORM is the instruction which describes the components of an object. Sym must be a valid symbol. Vall and val2 must be integer values ($1 \leqslant val \leqslant 32767$). They signify:

1. vall : length of the component in bytes,
2. val2 : displacement of the component with respect to an address. The address may be calculated during program execution, stored in a word-string and referenced in instruction GETADR.

<u>Examples:</u>

```
1.  F1  FORM  (IDEN, 8, 0)
2.      FORM  (CAT, 1, 0)
3.      FORM  (REF, 5, 6)
4.      FORM  (DATA, 76, 12)
5.  F2  FORM  (LENGTH, 12, 0)
6.  F3  FORM  (TITLE, 27)
```

The first instruction defines a terminal component named IDEN, with a length of 8 bytes and zero bytes displacement in respect to the start address.
The next four instructions may be described in the same way. The last instruction defines a terminal component with name TITLE and a length of 27 bytes.(In the internal representation, the part of the instruction, normally containing the displacement, is set to zero.) The field TITLE may be used in an instruction GETADR together with the parameter FIELD.

SYNTAX:

The format of the instruction is:

name	operation	operand
lab	SYNTAX	$\left\{\begin{matrix}\text{FORM}\\\text{DATA}\end{matrix}\right\}$

SYNTAX is an instruction which has to precede the first instruction of a syntax table.
lab must be a valid symbol. FORM is coded if the successive syntax table is the "format syntax", DATA if the syntax table is a "data syntax". Different data syntaxes are distinguished by different labels.
The sequence of the different syntax tables is only limited by rule No. 2 of chapter 4.

```
Examples:   SYN1    SYNTAX    DATA
                      :
                first data syntax
                      :
            SYN2    SYNTAX    DATA
                      :
                second data syntax
                      :
            SYNF    SYNTAX    FORM
                      :
                  format syntax
                      :
```

ERMESS:

The format of the instruction is:

name	operation	operand
lab or blank	ERMESS	val, const

ERMESS is the instruction which prepares a table of error messages coded by the user. These error messages may be used together with the instruction ERROR (semantic interpreter) to print a message in the case where an error condition arises during program execution. Val must be an integer value ($1 \leq$ val ≤ 127). It is referenced in the operand field of an instruction ERROR. Const. is a character constant representing the text to be printed.

Examples: 1. ERMESS 1,C'INCORRECT REFERENCE FIELD'
2. ERMESS 3,C'INVALID CHARACTER'

5. The SLC-Instructions of the Semantic Interpreter

This chapter describes the semantic instructions which are used to code the semantic action routines activated by the data or format syntax. An action routine is defined as a sequence of logically connected instructions, starting with a labelled one, up to the instruction RETUR. The label of the first instruction may be considered as the name attributed to the action routine.

CATEN:

The format of the instruction is:

name	operation	operand
lab or blank	CATEN	sr1, sr2

CATEN copies the string sr2 and adds it to the string srl. The string sr2 remains unchanged.
If the length of the combined strings would exceed the length of the reserved area of srl the message ***ER5*** (see Appendix No. 4) is printed and the analysis of the input is terminated.

Example: CATEN WORDSTR. STRINGl

It is assumed that WORDSTR and STRINGl represent the following strings:

WORDSTR : 'DOUBLE'
STRINGl : '-BEAM'

After execution of the instruction, the result is:

WORDSTR : 'DOUBLE-BEAM'
STRINGl : '- BEAM'

SUB:

The format of the instruction is:

name	operation	operand
lab or blank	SUB	srl, sr2

SUB reduces the string srl by the string sr2. The operation is performed if the following conditions are fulfilled:

1. The length (ℓ1) of the string srl must be equal or greater than the length (ℓ2) of sr2.
2. The ℓ2 rightmost characters of srl must agree with the characters of sr2.

If the first condition is violated, the message ***ER6*** is printed.

If the second conditions is not fulfilled, the message ***ER10*** is printed.
In both cases the analysis of the input is terminated.

Example: SUB WORDL, WORDSU

It is assumed that WORDL and WORDSU represent the following strings:

WORDL : 'TERMINATED'
WORDSU : 'INATED'

After execution of the instruction, the result is:
WORDL : 'TERM'
WORDSU : 'INATED'

SETSTR:

The format of the instruction is:

name	operation	operand
lab or blank	SETSTR	$\begin{Bmatrix} C \\ L \\ A \end{Bmatrix}$, sr1, sr2

If the letter C is coded, SETSTR copies the string sr2 in string sr1.
If the letter L is coded, SETSTR places the length attribute (integer value) of sr2 in string sr1.
If the letter A is coded, SETSTR places the address (leftmost byte) of string sr2 in string sr1.

Examples: SETSTR C, STRI1, STRI2
SETSTR L, STRI1, STRI2
SETSTR A, STRI1, STRI2

It is assumed that STRI2 represents the following string:

(STRI1 should be empty):

STRI2 : '31.12.72'

After execution of the first instruction, the result is:

STRI1 : '31.12.72'

STRI2 : '31.12.72'

Execution of the second instruction yields:

STRI1 : '8'(the value is coded in binary format)

STRI2 : '31.12.72'

Execution of the third instruction results in:

STRI1 : '4A8BC'(4A8BC is assumed to be the address of sr2)

STRI2 : '31.12.72'

TRANS:

The format of the instruction is:

name	operation	operand
lab or blank	TRANS	sr, tab

TRANS searches the string sr in a table tab, generated by instructions GENTAB and replaces it by the string which has been linked to it. If the string sr is not found in the table tab, the message ***ER7*** is printed and the analysis of the input is terminated.

<u>Example:</u> TRANS DSCSTR, TAB1

It is assumed that DSCSTR and TAB1 are represented by:

DSCSTR : 'S, BA'

TAB1

STRING	LINKED STRING
M	1A
B	NOT PRESENT
S, BA	3C
C, LB	52

After execution, the result is:

DSCSTR : '3C'

SORT:

The format of the instruction is:

name	operation	operand
lab or blank	SORT	$\left\{\begin{array}{l}\text{UP}\\\text{DOWN}\end{array}\right\}$, sr

The instruction SORT sorts the characters of a string. If UP is coded, the sort is performed in ascending binary weight, if DOWN is coded, in descending binary weight.

Example: SORT UP, STR1

It is assumed that STR1 is represented by:

STR1 : '4A21203C' (hexadecimal format, two digits represent a character)

After execution of the instruction, the result is:

STR1 : '20213C4A'

LOOKTAB:

The format of the instruction is:

name	operation	operand
lab or blank	LOOKTAB	sr, tab, sem

LOOKTAB searches a string sr in a table tab, which has been generated by GENTAB instructions. If the string is present, the program (interpreter) branches to the semantic instruction, labelled sem, given in the third operand. If the string is not present in the table, the program continues with the next sequential instruction.

```
Example:          LOOKTAB     DSCSTR, TAB1, PRETAB
                  SETSTR      C, WORDSTR, DSCSTR
                     :
                  RETUR       BACK
        PRETAB    CATEN       NUMBSTR, DSCSTR
                     :
                  RETUR       BACK
```

If the string represented by DSCSTR is found in table TAB1, the program branches to the instruction labelled PRETAB and continues execution till the instruction RETUR is encountered. If the string is not found in TAB1, the instructions SETSTR till RETUR are executed.

COMPSTR:

The format of the instruction is:

name	operation	operand
lab or blank	COMPSTR	sr1, sr2, sem

COMPSTR compares two strings character for character. If they are equal, the program (interpreter) branches to the semantic instruction, labelled sem, given in the third operand. If they are not equal or the length of the two strings is different, the program proceeds to the next sequential instruction.

```
Example:          COMPSTR     STR1,STR2,NOR4
                     :
                  RETUR       BACK
        NOR4      SEND        STR1
                     :
                  RETUR       BACK
```

If the character sequence of the strings STR1 and STR2 are equal, the program branches to NOR4, otherwise the next sequential instruction is executed.

SETWI:

The format of the instruction is:

name	operation	operand
lab or blank	SETWI	sr

SETWI declares a string as word item (a flag bit is set on).

Example: SETWI WORDSTR

SETNWI:

The format of the instruction is:

name	operation	operand
lab or blank	SETNWI	sr

SETNWI declares a string as non-word item (the flag bit is set off).

Example: SETNWI NUMBSTR

CLEAR:

The format of the instruction is:

name	operation	operand
lab or blank	CLEAR	sr

CLEAR deletes a string. The length attribute of the string is set zero (the reserved area of the string is not cleared).

Example: CLEAR DSCSTR

SEND:

The format of the instruction is:

name	operation	operand
lab or blank	SEND	sr

SEND stores a string sr according to its disposition.
If the string is a word item, it is stored in an internal table WORDS. At the end of the input analysis the content of WORDS may be saved on an output data set.
If the string is a non-word item, it is stored on an output data set named TEXTABLE.

In case of an empty string, the error message ***ER9*** is printed and the execution of the program is terminated.

<u>Example:</u> SEND DSCSTR

SETCNT:

The format of the instruction is:

name	operation	operand
lab or blank	SETCNT	sr, val

SETCNT assigns the value val to the string sr (4 bytes!). Val must be an integer between -32768 and +32767.

<u>Example:</u> SETCNT COUNT1, 5

After execution of the instruction 5 has been assigned to COUNT1.

ADDCNT:

The format of the instruction is:

name	operation	operand
lab or blank	ADDCNT	sr, val

ADDCNT adds the value val to the value already contained in the string sr (4 bytes!).

Example: ADDCNT COUNT1, 23

It is assumed that COUNT1 has the value 5. After execution of the instruction, the value of COUNT1 is 28.

TESTCNT:

The format of the instruction is:

name	operation	operand
lab or blank	TESTCNT	sr, val, sem

TESTCNT compares the contents of the string sr (4 bytes) with the value val and branches if the two values are equal to the instruction labelled sem, given in the third operand.
If the two values are not equal, the program proceeds with the next sequential instruction.

Example:

```
        TESTCNT  COUNT1, 10, AR15
           :
        RETUR    BACK
AR15    CATEN    STR1,STR2
           :
        RETUR    BACK
```

If the value of COUNT1 is 10, the interpreter branches to the instruction labelled AR15. If the value of COUNT1 is different from 10, the program proceeds to the next sequential instruction.

ONSW:

The format of the instruction is:

name	operation	operand
lab or blank	ONSW	sr

ONSW sets a switch in the on-state. Sr is a 4-byte string.

<u>Example:</u> ONSW SWIT1

OFFSW:

The format of the instruction is:

name	operation	operand
lab or blank	OFFSW	sr

OFFSW sets a switch in the off-state. Sr is a 4-byte string.

<u>Example:</u> OFFSW SWIT1

TESTSW:

The format of the instruction is:

name	operation	operand
lab or blank	TESTSW	sr, sem

TESTSW tests if the switch sr is in the on-state. If the answer is yes, the program branches to the instruction labelled sem. If the answer is no, the program proceeds to the next sequential instruction.

<u>Example:</u>

```
          TESTSW    SWIT1, ACTION12
            :
          RETUR     BACK
ACTION12  SEND      WORDSTR
            :
          RETUR     BACK
```

If SWIT1 is in the on-state, the program branches to the instruction labelled ACTION12. If SWIT1 is in the off-state, the interpreter proceeds to the next sequential instruction.

RETUR:

The format of the instruction is:

name	operation	operand
lab or blank	RETUR	$\begin{Bmatrix} \text{BACK} \\ \text{nod} \end{Bmatrix}$

RETUR must be the last instruction of a semantic action routine. If the operand BACK is coded, control is returned to the parser, which continues execution at the node of the syntax tree which is specified in the keyword parameter NEXT. If the operand nod is selected, the parser branches to the syntax instruction with label nod.

Examples:	RETUR	BACK
	RETUR	NODER11

ERROR:

The format of the instruction is:

name	operation	operand
symbol or blank	ERROR	val

ERROR causes printing of a message. Val must be an integer value ($1 < \text{val} \leqslant 127$). It specifies the code of the error message which has been selected.

The error messages have been defined with the instruction ERMESS.

If a selected error code does not exist, the error diagnostic ***ER8*** is printed.

Example: ERROR 5

The message associated to the error code 5 is printed.

TRACE:

The format of the instruction is:

name	operation	operand
lab or blank	TRACE	val, sr

TRACE is an instruction which may be used in debugging the syntax and the associated action routines. TRACE may be coded at any point of the semantic action routines. The operand val serves as identification number for the display. Val must be an integer $0 \leq val \leq 99$. The operand sr defines the string to be displayed.
When the instruction TRACE is executed, the following informations are displayed:

1. The identification number (val)
2. The name of the string
3. A 16 digit field describing the properties of the string. Two subfields are of interest:
   - digits 3, 4 contain the length of the string,
   - digits 8 to 16 contain the address of the string.
   (both fields are printed in hexadecimal format)
4. The string itself.

Example: TRACE 3 , STR1
TRACE 5 , WORDSTR

If the instruction is encountered, it displays the actual status of the strings STR1 and WORDSTR:

3	STR1	00060000000A0338	MEMBER
5	WORDSTR	000E0000000A14DA	ADMINISTRATION

PRISTRI:

The format of the instruction is:

name	operation	operand
lab or blank	PRISTRI	sr

PRISTRI is an instruction which is used on the semantic level to print the contents of a string.
If the string is empty, the message ***WA1*** is printed and execution continues. If however, the length of the string sr is greater than 132 bytes, the message ***ER11*** is printed and the job is terminated abnormally.

<u>Example:</u> PRISTRI STRIN1

APPENDIX No. 2

The appendix contains the following information:
(valid only for the SLC-II system on the IBM 370/165 at Ispra)

1. The list of the control statements to create the load module from the user's program
2. The list of control statements for the execution of TEXTANAL (no restart)
3. The list of control statements if the restart facility is used.

```
//          EXEC    AHCL  US=SLC2, UMC=3330, VMC= PERS00
//CMP.SYSIN    DD    *
                     .
                     .
            user program
                     .
                     .
/*                   .
//LKED.SYSLMOD   DD   DSN = SLC2.LIBRARY.SYSTEM,
//          UNIT=3330, VOL = SER = PERSOO, DISP =(OLD, KEEP, KEEP)
//LKED.SYSIN       DD *
      NAME       INGRAM
/*
```

Fig. 1 - The control statements for the load module creation. The load module is named INGRAM and added with this member-name to the data set SLC2, LIBRARY.SYSTEM.

```
// JOBLIB   DD  DSN=SLC2.LIBRARY.SYSTEM,
            UNIT=3330, VOL=SER=PERSOO, DISP=(OLD, KEEP, KEEP)

// GO       EXEC  PGM=SLCMONIT

// GO.SYSOUT   DD  SYSOUT=A, DCB=(RECFM=FA, LRECL=132, BLKSIZE=
                                                        132)
// GO.TEXTABLE   DD   DSN=SLCII.TEXTABLE, UNIT=SYSSQ,
//                           SPACE=(CYL, 5), DISP=(NEW, DELETE),
//                           DCB=(RECFM=F, LRECL=7200,
                                  BLKSIZE=7200)

// GO.WORDTAB   DD   DSN=SLCII.WORDTABL, UNIT=SYSSQ,
//                      SPACE=(CYL, 2), DISP=(NEW, DELETE),
//                      DCB=(RECFM=F,  LRECL=7200, BLKSIZE=
                                                   7200)

// GO.WORDFILE   DD   DSN=SLCII.WORDFILE, UNIT=SYSSQ,
//                      SPACE=(CYL, 3), DISP=(NEW, DELETE),
//                      DCB=(RECFM=F, LRECL=7200, BLKSIZE=
                                                   7200)
// GO.FREQTAB   DD   DSN=SLCII.FREQTAB, UNIT=SYSSQ,
//                      SPACE=(CYL, 2), DISP=(NEW, DELETE),
//                      DCB=(RECFM=F, LRECL=7200, BLKSIZE=
                                                   7200)
// GO.FREQFILE   DD   DSN=SLC2.FREE1, UNIT=3330,
//                      VOL=SER=PERSOO, DISP=(OLD, KEEP),
//                      DCB=(RECFM=FB, LRECL=36, BLKSIZE=7200)

// GO.SYSIN        DD  *
GRAMMAR  INGRAM
OPTIONS     0C0F0000
/*

//GO.INPUTEXT   DD  *
                     :
                 input text
                     :
/*
```

Fig. 2 - The control statements for the execution of TEXTANAL. In the present example the data sets with the DDnames TEXTABLE, WORDTAB, WORDFILE, and FREQTAB are defined as scratch data sets. Only FREQFILE is kept after program execution. The DDnames of all data sets and the DCB parameter of FREQFILE must be coded as shown. The other DCB blocks may be changed.

The cards GRAMMAR and OPTIONS are input cards for the monitor program SLCMONIT. GRAMMAR specifies the user load module to be used with TEXTANAL. OPTIONS sets flags used by SLCMONIT and TEXTANAL (see chapter 3). The names GRAMMAR, OPTIONS start in column 1, the second field (INGRAM, 0C0F0000) in column 16.

```
// JOBLIB    DD    DSN=SLC2.LIBRARY.SYSTEM,
                           .
                           .
            same as specified in Fig. 2
                           .
                           .
// GO.SYSIN  DD  *
GRAMMAR  INGRAM
OPTIONS    0C0F0000
TISTART    00034133  0032
/*
// GO.INPUTEXT  DD  DSN=SLC2.OUTPUT.CID, UNIT=3330,
//                  VOL=SER=PERSOO,DISP=(OLD,KEEP,KEEP),
//                  DCB=(RECFM=F,LRECL=210,BLKSIZE=210)
/*
```

Fig. 3 - The control statements for the execution of TEXTANAL using the restart facility.
The explanation given for Fig. 2 is also valid in this case. The only difference is the monitor control card TISTART. The first value (00034133) specifies the number of records to be skipped, the second value (0032) specifies the displacement where analysis restarts (00034133 is coded at column 16 to 23; 0032 from column 25 to column 28) (for limitations, see chapter 4).

APPENDIX No. 3

Assembly Time Diagnostics of the Users Coded Program

ER1 THE NAME**WATOM** MAY NOT BE USED

Explanation:

The name WATOM is reserved for internal use.

Action:

Choose an other name.

ER2PREVIOUSLY DEFINED--NNN

Explanation:

The name NNN has already been used in an instruction GENSTR, FORM, INSCAN, MULTSCAN or as a label.

Action:

Choose an other name.

ER3ILLEGAL DEFINITION--NNN

Explanation:

The definition NNN is not valid. If NNN is a name of a string, the definition contains more than two elements e.g. (STRING1, 244, 5).

Action:

Supply correct definition.

ER4 NAME IS MISSING

Explanation:

1. The first instruction GENTAB, FORM, INSCAN(MULTSCAN) does not contain a label,
2. An instruction SYNTAX does not contain a label.

Action:

Code a label.

ER5 LENGTH INCORRECT--NNN

Explanation:

Only one character of the source text alphabet may be specified.

Action:

Correct error.

ER6 ILLEGAL DATA FORMAT--NNN

Explanation:

The format of the data NNN is invalid. Allowed formats are:

1. Hexadecimal data constants e.g. X'AB145'
2. Binary data constants e.g. B'00110110'
3. Character constants e.g. C'AB14C$'

Action:

Supply correct data format.

ER7 QUOTES ARE MISSING--NNN

Explanation:

The expression NNN has to be enclosed in quotes.

Action:

Correct error.

ER8 ILLEGAL RELATIONAL OPERATOR--NNN

Explanation:

The expression NNN is an invalid relational operator. The valid operators are:

EQ, GE, GT, LE, LT, NE

Action:

Supply valid relational operator.

ER9 UNDEFINED OPERAND--NNN

Explanation:

The operand NNN in an instruction PARSE is not defined.

Action:

Define operand by means of the instructions GENSTR or FORM.

ER10 KEYWORD PARAMETERS MISSING--NNN

Explanation:

No keyword parameters have been specified in an instruction PARSE.

Action:

Specify at least one keyword parameter.

ER11 KEYWORD OUT OF RANGE--&ACT= NNN

Explanation:

The program internal value attributed to the keyword ACT exceeds 255.

Action:

Reduce the number of action routines.

ER12 ILLEGAL CHARACTERS--NNN

Explanation:

The expression NNN contains illegal character(s). The allowed characters are A-Z, $, #, 1-9, blank.

Action:

Correct error.

ER13 KEYWORD -- ADR MISSING

Explanation:

The keyword ADR= in an instruction INPUT or GETADR is missing.

Action:

Supply keyword parameter.

ER14 STRING--NNN NOT DEFINED

Explanation:

The string NNN does not exist.

Action:

Correct typing error, or

generate string with instruction GENSTR.

ER15 KEYWORD -- STRING MISSING

Explanation:

The keyword STRING= in an instruction OUTPUT is missing.

Action:

Supply keyword parameter.

ER16 PARAMETER COMP = NNN NOT DEFINED

Explanation:

The keyword parameter NNN, which should be a label of an instruction FORM, does not exist.

Action:

Correct typing error, or generate the parameter.

ER17 ONE KEYWORD ONLY

Explanation:

Two keyword parameters have been coded in the operand field of GETADR.

Action:

Correct error.

ER18 KEYWORDS ARE MISSING

Explanation:

No keyword parameter has been coded in the operand field of GETADR.

Action:

Supply one keyword parameter.

ER19 KEYWORD -- OLD MISSING

Explanation:

The keyword parameter OLD = of the instruction VAL is missing.

Action:

Supply the keyword parameter.

ER20 KEYWORD -- NEW MISSING

Explanation:

The keyword parameter NEW = of the instruction VAL is missing.

Action:

Supply the keyword parameter.

ER21 KEYWORD -- INCR MISSING

Explanation:

The keyword parameter INCR = of the instruction VAL is missing.

Action:

Supply the keyword parameter.

ER22 KEYWORD -- SCAN MISSING

Explanation:

The keyword parameter SCAN = of the instruction ANALYSE is missing.

Action:

Supply the keyword parameter.

ER23 KEYWORD -- SYNTAX MISSING

Explanation:

The keyword parameter SYNTAX = of the instruction ANALYSE is missing.

Action:

Supply the keyword parameter.

ER24 KEYWORD -- TEXT MISSING

Explanation:

The keyword parameter TEXT = of the instruction ANALYSE is missing.

Action:

Supply the keyword parameter.

ER25 KEYWORD SCAN = NNN NOT DEFINED

Explanation:

The name NNN of a scanner table does not exist.

Action:

Correct typing error or generate a scanner table with name NNN.

ER26 KEYWORD SYNTAX = NNN NOT DEFINED

Explanation:

The name NNN of a syntax table does not exist.

Action:

Correct typing error or generate the name of the syntax table with the instruction SYNTAX.

ER27 KEYWORD TEXT = NNN NOT DEFINED

Explanation:

The name NNN representing a terminal component or a string, does not exist.

Action:

Correct typing error or generate the name of the terminal component with the instruction FORM or GENSTR.

ER28 SYNTAX TYPE NOT SPECIFIED

Explanation:

The syntax type (FORM or DATA) has not been coded in the operand field of an instruction SYNTAX.

Action:

Correct error.

ER29 INVALID SYNTAX TYPE

Explanation:

The parameter coded in the operand field of an instruction SYNTAX is not valid. The allowed parameters are FORM, DATA.

Action:

Correct error.

ER30 ERROR CODE ** NNN ** IS NOT VALID

Explanation:

NNN must be an integer between 1 and 127.

Action:

Correct error.

ER31 THE INTEGER **NNN** IS NOT VALID

Explanation:

NNN must be within the following range $-32768 \leq NNN \leq 32767$

Action:

Correct error.

ER32 KEYWORD COD IS MISSING

Explanation:

The keyword parameter COD = has not been defined in an instruction TERM.

Action:

Code the parameter COD =.

ER33 POSITIONAL PARAMETER(S) MISSING

Explanation:

One or both positional keyword parameters of an instruction NOTERM are missing. The two parameters are mandatory.

Action:

Correct error.

ER34 TABLE -- NNN NOT DEFINED

Explanation:

The table with name NNN does not exist.

Action:

Correct typing error or generate the table with the instruction GENTAB.

ER35 ILLEGAL OPTION -- NNN

Explanation:

The option specified in an instruction SETSTR or SORT is wrong.

Action:

Supply correct option.

ER36 ILLEGAL INTEGER -- NNN

Explanation:

The integer NNN must be within the range: $0 \leqslant$ NNN $\leqslant 99$ (instruction TRACE)

Action:

Correct error.

ER37 ILLEGAL FORMAT

Explanation:

The error message defined in the operand field of an instruction ERMESS has to be coded as a character constant (C 'NNNN').

Action:

Correct error.

ER38 ACTION ROUTINE ACT=NNN NOT DEFINED

Explanation:

The action routine named NNN in an instruction PARSE, TERM, NOTERM, BRUNC does not exist.

Action:

Correct typing error or code an action routine with the name NNN.

APPENDIX No. 4

Execution-time Diagnostics of TEXTANAL

ER1 AVAILABLE MEMORY TOO SMALL

Explanation:

The memory allocated to this job is too small in order to contain all the buffers and the internal program tables.

Action:

Allocate more memory to the job and repeat execution.

ER2 INVALID END OF FILE

Explanation:

The last record of the input text terminated with a word atom. The program tried to concatenate the atom with the next record, which does not exist.

Action:

The last atom of the input text must be an individual or multiple atom.

ER3 OVERFLOW IN INDEX-STACK

Explanation:

The number of nested NOTERM instructions is greater than 20.

Action:

Reduce in the syntax the number of nested NOTERM instructions.

ER4 THE INDEX-STACK IS EMPTY

Explanation:

TEXTANAL performed a branch from a subtree back into the main syntax tree, but the index stack was empty. Error in data syntax.

Action:

Correct error.

ER5 STRING OVERFLOW

CCCCC (1. string)

CCCCC (2. string)

Explanation:

The length of the string, which would result from the execution of an instruction CATEN, would be greater than 255 bytes. The instruction is not executed. Both strings or the first 132 bytes are printed and the job is terminated. Attention: the strings may contain unprintable characters.

Action:

Change syntax and/or the semantic action routines.

ER6 STRING UNDERFLOW

CCCCC (1. string)

CCCCC (2. string)

Explanation:

The length of the second string in an instruction SUB is greater than the length of the first one. The instruction is not executed. Both strings or the first 132 bytes are printed and the job is terminated. Attention: the strings may contain unprintable characters.

Action:

Change syntax and/or semantic action routines.

ER7 TABLE ITEM DOES NOT EXIST -- NNN

Explanation:

An instruction TRANS could not find the item NNN in the table. Attention: NNN may consist of unprintable characters.

Action:

Insert item or change syntax and/or semantic action routines. Debug with TRACE.

ER8 ERROR CODE DOES NOT EXIST -- NNN

Explanation:

An error code specified in an instruction ERROR does not exist. NNN is the error code number.

Action:

Correct typing error or generate the error code in ERMESS.

ER9 EMPTY STRING IN SEND INSTRUCTION -- NNN

Explanation:

The string in an instruction SEND has length zero. NNN is the name of the string.

Action:

Change syntax and/or semantic action routines. Use instruction TRACE for debugging.

ER10 NO CHARACTER AGREEMENT IN SUB

CCCCC (1. string)

CCCCC (2. string)

Explanation:

The rightmost characters of the first string in an instruction SUB do not agree with the characters of the second string. Both strings or the first 132 bytes are printed and the job is terminated.

Action:

Change syntax and/or semantic action routines. Use instruction TRACE for debugging.

ER11 STRING TOO LONG FOR PRINTER BUFFER

CCCCC (string)

Explanation:

The instructions OUTPUT or PRISTRI allow only the printing of strings less than or equal to 132 bytes. The first 132 bytes are printed and the execution of the program is terminated.

Action:

Change syntax and/or semantic action routines. Use the instruction TRACE for debugging.

ER12 CONCATENATION BUFFER OVERFLOW

CCCCC (first 50 bytes of buffer)

Explanation:

An input record terminated with a word atom. The next record is concatenated to the word atom (see also Appendix 1,chapter 2). The resulting length would exceed the length of the reserved core area (500 bytes). The word atom and the first 50 bytes of the next record are printed and the job is terminated.

Action:

Reduce length of input record. If this is not possible, the core area for concatenation has to be increased.

ER13 LENGTH OF STRINGS IN PARSE NOT EQUAL

Explanation:

The length of the strings to be compared in the positional parameter in PARSE are not equal. Correct coding is:

1. the string must be a word-string (4 bytes) if it is compared with a) numerical value,
   b) switch.

2. the strings must be of equal length, if two strings are compared.

Action:

Correct syntax.

ER14 LENGTH OF DATA FIELD IS ZERO

Explanation:

A terminal component, specified in an instruction ANALYSE, has length zero.

Action:

Correct error in the instruction FORM.

ER15 SYNTAX ERROR NO. NNN

Explanation:

The parser found the last instruction of a syntax node without being able to match the scanned atom. NNN is the value coded in the keyword parameter ERC. It allows to locate the error in the syntax tree.

Action:

Correct data syntax.

ER16 ILLEGAL BRANCH IN INSTRUCTION - RETUR -

Explanation:

The label of the syntax node coded in the operand field of an instruction RETUR does not belong to the data syntax being used.

Action:

Supply correct label.

WA1 EMPTY STRING TO PRINTER

Explanation:

The instructions OUTPUT or PRISTRI want to print a string of length zero.

Action:

Correct syntax and/or semantic action routines.

APPENDIX No. 5

Appendix No. 5 gives as an example a user coded program and the results.

The input text is assumed to be coded on 80 column cards or stored in card image format (LRECL = 80).

The "object" to be analyzed is in this example the card or the 80 byte record respectively. It consists of three components, which are also terminal components.

1. component CAT (card column 1)
2. component REF (card columns 2 to 8)
3. component DATA (card columns 9 to 70)

THE USER PROGRAMM

```
  3          GENSTR SW4,SW5,SWI1,TRASTR
 73          GENSTR ADBUF
 89          GENSTR NO30,NO32,NC11
126          GENSTR (WCAT,1),(WREF,7),(WIDEN,8)
163          GENSTR (STR1,255),(STR2,255),(STR3,255)
200          GENSTR (CODESTR,255),(WORDSTR,255),(CODESTRX,255)
237          GENSTR (DSCSTRX,255),(COORSTRX,255),(DSCSTR,255)
274          GENSTR (COORSTR,255),(LISTR,255)
299          GENSTR (LPAR,C'('),(RPAR,C')'),(DOLLSTR,C'$')
336          GENSTR (HYPHENS,C'(HYPHEN)'),(HYPHENC,X'4A')
361          GENSTR (SHARPSTR,C'='),(RIFSTR,C'==RIF==')
386          GENSTR (CATSTR,C'=$CAT=$'),(ENDSTR,C'===')
411          GENSTR (BLANKL,CL122' '),(BLANKCH,C' ')
```

```
437 WIDTAB    GENTAB (C'M',X'10')
453           GENTAB (C'N',X'15')
461           GENTAB (C'S',X'1A')
469           GENTAB (C'BA',X'20')
477           GENTAB (C'E1',X'21')
485           GENTAB (C'I1',X'22')
493           GENTAB (C'MI',X'25')
501           GENTAB (C'SL',X'2A')
509           GENTAB (C'M,SL',X'45')
517           GENTAB (C'S,BA',X'4A')
525           GENTAB (C'S,SL',X'50')
533           GENTAB (X'102A',C'M,SL')
541           GENTAB (X'1A20',C'S,BA')
549           GENTAB (X'1A2A',C'S,SL')
557 NWIDTAB   GENTAB (C'B',X'1')
568 NWIDTAB   GENTAB (C'TC',X'30')
576 NWIDTAB   GENTAB (C'TG',X'35')
584 NWIDTAB   GENTAB (C'TR',X'40')
592 NWIDTAB   GENTAB (C'B,E1',X'42')
600           GENTAB (C'B,I1',X'43')
608           GENTAB (X'121',C'B,E1')
616 NWIDTAB   GENTAB (X'122',C'B,I1')
624 LDTAB     GENTAB C'E',C'F',C'D'
643 CWIDTAB   GENTAB
650 CATTAB    GENTAB (C'4',C'001')
661 CATTAB    GENTAB (C'7',C'002')
```

```
670 SCAN1     INSCAN (\$,C'\$')
688           INSCAN (CUMMA,C',')
696           INSCAN (=,C'=')
704           INSCAN (HYPHEN,C' ')
712           MULTSCAN (MBLANK,C' ')
```

```
721          ERMESS 3,C'STRING DISCARDED - SKIP TO NEXT BLANK.'
730          ERMESS 4,C'INCORRECT DESCR. STRING - SKIP TO NEXT BLANK.'
738          ERMESS 5,C'INCORRECT LITERAL DESC. - SKIP TO NEXT $BLANK.'
```

```
747 F1          FORM   (IDEN,8,0)
771             FORM   (CAT,1,0)
789             FORM   (REF,7,1)
807             FORM   (DATA,62,8)
```

```
 826 AR01      SETSTR C,WCAT,CAT
 844           SETSTR C,WREF,REF
 850           SETSTR C,WIDEN,IDEN
 856           SETWI RIFSTR
 862           SEND  RIFSTR
 868           CLEAR LISTR
 870           CATEN LISTR,BLANKL
 876           PRISTRI LISTR
 882           PRISTRI LISTR
 888           CLEAR LISTR
 890           CATEN LISTR,RIFSTR
 896           CATEN LISTR,REF
 902           PRISTRI LISTR
 908           CLEAR LISTR
 910           CATEN LISTR,CATSTR
 916           SETSTR C,TRASTR,CAT
 922           TRANS TRASTR,CATTAB
 928           CATEN LISTR,TRASTR
 934           PRISTRI LISTR
 940           SETNWI REF
 946           SEND  REF
 952           SETWI CATSTR
 958           SEND  CATSTR
 964           SETNWI TRASTR
 970           SEND  TRASTR
 976           ONSW  SWI1
 982           RETUR BACK
 987 AR02      SETSTR C,WCAT,CAT
 997           SETSTR C,WIDEN,IDEN
1003           CLEAR LISTR
1005           CATEN LISTR,CATSTR
1011           SETSTR C,TRASTR,CAT
1017           TRANS TRASTR,CATTAB
1023           CATEN LISTR,TRASTR
1029           PRISTRI LISTR
1035           SETWI CATSTR
1041           SEND  CATSTR
1047           SETNWI TRASTR
1053           SEND  TRASTR
1059           RETUR BACK
1064 AR03      TESTSW N011,AR031
1074           TESTSW N030,AR032
1080           TESTSW N032,AR033
1086           RETUR BACK
1091 AR031     RETUR NODER11
1100 AR032     RETUR NODE30
1109 AR033     RETUR NODE32
1118 AR1       ONSW  SW4
1128           RETUR BACK
1133 AR2       CATEN STR1,WATOM
1143           RETUR BACK
1148 AR3       SEND  STR1
1158           RETUR BACK
1163 AR4       CATEN STR1,WATOM
1173           SETSTR C,STR2,STR1
```

```
1179            RETUR BACK
1184 AR5        CLEAR CODESTR
1190            CLEAR WORDSTR
1192            OFFSW SW4
1198            RETUR BACK
1203 AR64       TRANS STR1,NWIDTAB
1213            CATEN CODESTRX,STR1
1219            RETUR BACK
1224 AR61       SETNWI  WORDSTR
1234            ONSW   SW5
1240 AR62       CATEN DSCSTRX,WATOM
1250            SETSTR C,STR1,WATOM
1256            TESTSW SW5,AR64
1262            TRANS STR1,WIDTAB
1268            CATEN CODESTRX,STR1
1274            RETUR BACK
1279 AR6        CLEAR COORSTR
1285            CLEAR DSCSTR
1287            CLEAR DSCSTRX
1289            CLEAR CODESTRX
1291            SETSTR L,STR1,WORDSTR
1297            CATEN COORSTR,LPAR
1303            CATEN COORSTR,STR1
1309 AR63       OFFSW SW5
1319            LOOKTAB  WATOM,WIDTAB,AR62
1325            LOOKTAB  WATOM,NWIDTAB,AR61
1331            ERROR 4
1337            RETUR ER11
1342 AR7        CATEN DSCSTRX,WATOM
1352            RETUR BACK
1357 AR82       TRANS DSCSTRX,NWIDTAB
1367            CATEN CODESTR,DSCSTRX
1373            RETUR BACK
1378 AR81       CATEN DSCSTR,DSCSTRX
1388            TESTSW SW5,AR82
1394            TRANS DSCSTRX,WIDTAB
1400            CATEN CODESTR,DSCSTRX
1406            RETUR BACK
1411 AR812      TRANS DSCSTRX,NWIDTAB
1421            SETSTR C,DSCSTR,DSCSTRX
1427            TRANS DSCSTRX,NWIDTAB
1433            CATEN CODESTR,DSCSTRX
1439            RETUR BACK
1444 AR811      SETSTR C,DSCSTRX,CODESTRX
1454            TESTSW SW5,AR812
1460            TRANS DSCSTRX,WIDTAB
1466            SETSTR C,DSCSTR,DSCSTRX
1472            TRANS DSCSTRX,WIDTAB
1478            CATEN CODESTR,DSCSTRX
1484            RETUR BACK
1489 AR8        SORT   UP,CODESTRX
1499            OFFSW SW5
1505            LOOKTAB CODESTRX,WIDTAB,AR811
1511            ONSW   SW5
1517            LOOKTAB CODESTRX,NWIDTAB,AR811
```

```
1523            ERROR 4
1529            RETUR ER11
1534 AR9        SETSTR L,STR1,WATOM
1544            CATEN WORDSTR,WATOM
1550            CATEN COORSTR,STR1
1556            CATEN COORSTR,RPAR
1562            CLEAR DSCSTRX
1564            CATEN DSCSTRX,DOLLSTR
1570            CATEN DSCSTRX,DSCSTR
1576            CATEN DSCSTRX,DOLLSTR
1582            SEND  DSCSTRX
1588            SETNWI COORSTR
1594            SEND  COORSTR
1600            RETUR BACK
1605 AR101      SEND  WORDSTR
1615            OFFSW SW4
1621            RETUR BACK
1626 AR102      SORT  UP,CODESTR
1636            SETWI WORDSTR
1642            LOOKTAB CODESTR,CWIDTAB,AR101
1648            SETNWI WORDSTR
1654            SEND  WORDSTR
1660            OFFSW SW4
1666            RETUR BACK
1671 AR10       TESTSW SW4,AR102
1681            SEND  WORDSTR
1687            RETUR BACK
1692 AR11       SETWI HYPHENS
1702            SEND  HYPHENS
1708            SETSTR C,STR3,STR2
1714            SUB   STR3,HYPHENC
1720            CATEN STR2,STR1
1726            CATEN STR3,STR1
1732            SETWI STR2
1738            SETWI STR3
1744            SEND  STR2
1750            SEND  STR3
1756            RETUR BACK
1761 AR161      CLEAR WORDSTR
1767            SETNWI WORDSTR
1773            SETSTR C,DSCSTR,SHARPSTR
1779            SETNWI DSCSTR
1785            CATEN DSCSTR,WATCM
1791            CATEN DSCSTR,SHARPSTR
1797            CLEAR COORSTR
1799            SETNWI COORSTR
1805            CATEN COORSTR,LPAR
1811            SETSTR L,STR1,WCRDSTR
1817            CATEN COORSTR,STR1
1823            RETUR BACK
1828 AR16       LOOKTAB WATOM,LDTAB,AR161
1838            ERROR 5
1844            RETUR ER9
1849 AR17       CATEN WORDSTR,WATOM
1859            RETUR BACK
```

```
1864 AR18      SUB    WORDSTR,DOLLSTR
1874           SETSTR L,STR1,WCRDSTR
1880           CATEN  COORSTR,STR1
1886           CATEN  COORSTR,RPAR
1892           SEND   DSCSTR
1898           SEND   WORDSTR
1904           OFFSW  NO30
1910           RETUR  BACK
1915 AR19      CLEAR  STR1
1921           RETUR  BACK
1926 AR21      ERROR  5
1936           ONSW   NO32
1942           RETUR  BACK
1947 AR22      ERROR  3
1957           ONSW   NO11
1963           RETUR  BACK
1968 AR26      SEND   STR1
1978           CLEAR  STR1
1980           CLEAR  CODESTR
1982           CLEAR  WORDSTR
1984           OFFSW  SW4
1990           RETUR  BACK
1995 AR27      ERROR  4
2005           ONSW   NO11
2011           OFFSW  SW4
2017           RETUR  BACK
2022 AR28      OFFSW  NO11
2032           RETUR  BACK
2037 AR29      ONSW   NO30
2047           RETUR  BACK
```

```
SYN2     SYNTAX DATA
NODE01   BRUNC  NEXT=NODE0,ACT=AR03
NODE0    TERM   NEXT=NODE16,COD=MBLANK
         BRUNC  NEXT=NODE16
NODE16   BRUNC  ACT=AR19,NEXT=NODE17
NODE17   TERM   ACT=AR2,NEXT=NODE18,COD=WORD
         TERM   NEXT=NODE19,COD=$
         TERM   NEXT=NODE20,COD==
         NOTERM NODE18,CHS
NODE18   TERM   ACT=AR4,NEXT=NODE36,COD=HYPHEN
         TERM   ACT=AR3,NEXT=NODE16,COD=MBLANK
         TERM   ACT=AR26,NEXT=NODE21,COD=$
         NOTERM NODE16,CWS,ACT=AR3
         BRUNC  ACT=AR22,NEXT=NODER11
NODE21   TERM   ACT=AR6,NEXT=NODE22,COD=WORD
         BRUNC  ACT=AR27,NEXT=NODER11
NODE22   TERM   ACT=AR7,NEXT=NODE25,COD=COMMA
         TERM   ACT=AR81,NEXT=NODE23,COD=$
         BRUNC  ACT=AR27,NEXT=NODER11
NODE25   TERM   ACT=AR63,NEXT=NODE26,COD=WORD
         BRUNC  ACT=AR27,NEXT=NODER11
NODE26   TERM   ACT=AR8,NEXT=NODE23,COD=$
         TERM   ACT=AR7,NEXT=NODE25,COD=COMMA
         BRUNC  ACT=AR27,NEXT=NODER11
NODE23   TERM   ACT=AR9,NEXT=NODE24,COD=WORD
         BRUNC  ACT=AR27,NEXT=NODER11
NODE24   TERM   ACT=AR1,NEXT=NODE21,COD=$
         TERM   ACT=AR10,NEXT=NODE16,COD=MBLANK
         BRUNC  ACT=AR27,NEXT=NODER11
NODER11  TERM   NEXT=NODER11,COD=COMMA
         TERM   NEXT=NODER11,COD=$
         TERM   NEXT=NODER11,COD==
         TERM   NEXT=NODER11,COD=WORD
         TERM   NEXT=NODER11,COD=HYPHEN
         TERM   NEXT=NODE16,COD=MBLANK,STOP=1,ERC=1,ACT=AR28
NODE19   TERM   NEXT=NODE20,COD=$
         BRUNC  ACT=AR5,NEXT=NODE21
NODE20   TERM   ACT=AR16,NEXT=NODE28,COD=WORD
         BRUNC  ACT=AR21,NEXT=NODE32
NODE28   TERM   NEXT=NODE29,COD=$
         TERM   NEXT=NODE30,COD==,ACT=AR29
         BRUNC  ACT=AR21,NEXT=NODE32
NODE29   TERM   NEXT=NODE30,COD=$,ACT=AR29
         BRUNC  ACT=AR21,NEXT=NODE32
NODE30   TERM   ACT=AR17,NEXT=NODE30,COD=MBLANK
         TERM   ACT=AR17,NEXT=NODE30,COD=HYPHEN
         TERM   ACT=AR17,NEXT=NODE30,COD=COMMA
         TERM   ACT=AR17,NEXT=NODE30,COD==
         TERM   ACT=AR17,NEXT=NODE30,COD=WORD
         TERM   ACT=AR17,NEXT=NODE35,COD=$,STOP=1,ERC=2
NODE35   TERM   ACT=AR18,NEXT=NODE16,COD=MBLANK
         BRUNC  NEXT=NODE30
NODE32   TERM   NEXT=NODE32,COD=WORD
         TERM   NEXT=NODE32,COD==
         TERM   NEXT=NODE32,COD=MBLANK
```

```
                 TERM   NEXT=NODE32,COD=HYPHEN
                 TERM   NEXT=NODE32,COD=COMMA
                 TERM   NEXT=NODE33,COD=$,STOP=1,ERC=3
       NODE33    TERM   NEXT=NODE16,COD=MBLANK
                 BRUNC  NEXT=NODE32
       NODE36    TERM   ACT=AR19,NEXT=NODE37,COD=MBLANK
                 TERM   ACT=AR4,NEXT=NODE18,COD=HYPHEN
                 BRUNC  ACT=AR22,NEXT=NODER11
       NODE37    NOTERM  NODE38,CHS
                 BRUNC  NEXT=NODE38
       NODE38    TERM   ACT=AR2,NEXT=NODE39,COD=WORD
                 BRUNC  ACT=AR22,NEXT=NODER11
       NODE39    TERM   ACT=AR11,NEXT=NODE16,COD=MBLANK
                 BRUNC  ACT=AR22,NEXT=NODER11
       CHS       TERM   ACT=AR2,NEXT=NODE43,COD=COMMA
                 TERM   ACT=AR2,NEXT=NODE43,COD=HYPHEN
                 BRUNC  EXIT=1
       NODE43    TERM   ACT=AR2,NEXT=NODE43,COD=COMMA
                 TERM   ACT=AR2,NEXT=NODE43,COD=HYPHEN
                 BRUNC  EXIT=1
       CWS       TERM   ACT=AR2,NEXT=NODE45,COD=COMMA
                 TERM   ACT=AR2,NEXT=NODE45,COD=WORD
                 BRUNC  TYPE=3
       NODE45    TERM   ACT=AR2,NEXT=NODE45,COD=COMMA
                 TERM   ACT=AR2,NEXT=NODE45,COD=WORD
                 TERM   NEXT=NODE46,COD=MBLANK
                 BRUNC       TYPE=3
       NODE46    BRUNC  EXIT=1
01718  ER11      EQU    NODER11
01774  ER9       EQU    NODE32
```

```
SYN1    SYNTAX FORM
P1      INPUT ADR=ADBUF
        GETADR ADR=ADBUF,COMP=F1
        PARSE 'SWI1 EQ CN',NEXT=P2
        PARSE ACT=AR01,NEXT=P3
P2      PARSE 'WIDEN EQ IDEN',NEXT=P3
        PARSE 'WCAT EQ CAT',NEXT=P3,ACT=AR01             NEW REF
        PARSE 'WREF EQ REF',NEXT=P3,ACT=AR02             NEW CAT
        PARSE ACT=AR01                                   NEW REF
P3      OUTPUT STRING=INPUT
        ANALYSE SCAN=SCAN1,SYNTAX=SYN2,TEXT=DATA
        PARSE NEXT=P1
        END
```

THE RESULTS

```
THE FOLLOWING OPTIONS WERE CHOSEN
NUMCYCLE       01
GRAMMAR        GRA1
OPTIONS        000F0000
```

```
==RIF==1812270
=$CAT=$001
41812270$S$IMPROVEMENTS $S$IN $S$OR $S$RELATING $S$TO $S$THE          NSA010
41812270$S$RECOVERY $S$OF $S$CAESIUM .                                NSA020/
=$CAT=$002
71812270$S$A METHOD IS GIVEN FOR SEPARATING CESIUM FROM AN            NSA030
71812270AQUEOUS SOLUTION , ESPECIALLY $M$CS$E1$137 FROM A             NSA040
71812270SOLUTION OF FISSION PRODUCTS . $M$THE METHOD                  NSA050
71812270COMPRISES CONTACTING THE SOLUTION WITH A                      NSA060
71812270WATER-IMMISCIBLE SOLVENT FOR CESIUM POLYBROMIDES IN THE       NSA070
71812270PRESENCE OF A BROMIDE SALT AND BROMINE , WHEREBY THE          NSA080
71812270CESIUM FORMS POLYBROMIDES AND IS EXTRACTED , AND              NSA090
71812270REMOVING THE EXTRACTED CESIUM FROM THE SOLVENT                NSA100
71812270BY STEAM STRIPPING OF THE FREE BROMINE . $M$THE               NSA110
71812270SOLVENT PREFERABLY IS NITROBENZENE WITH A HEAVY INERT         NSA120
71812270DILUENT ( $S$C$M$ER$I1$4 OR $S$C$I1$2$S$H$I1$2$M$BR$I1$4 )    NSA130
71812270ADDED TO INCREASE ITS DENSITY . $M$THE METHOD IS              NSA140
71812270HIGHLY SPECIFIC TO CESIUM   AT 90( EXTRACTION ,               NSA150
71812270ONLY 10( OF THE RUBIDIUM IS EXTRACTED AND LESS THAN 1(        NSA160
71812270OF ANY OF THE OTHER METALS . $S$A FLOW SHEET IS               NSA170
71812270INCLUDED . ( $S$D.L.C. ) .                                    NSA180/

==RIF==1812519
=$CAT=$001
41812519$S$ISOTOPIC $S$SEPARATION $S$AS $S$A $S$FACTOR                NSA010
41812519$S$IN $S$THE $S$SELECTION $S$OF $S$CANDIDATE                  NSA020
41812519$S$HEAT-SOURCE $S$RADIOISOTOPES .                             NSA030/
=$CAT=$002
71812519$S$A RE-EXAMINATION IS MADE OF THE RADIOISOTOPES WHICH        NSA040
71812519WERE CLASSIFIED AS REQUIRING ISOTOPIC SEPARATION IN           NSA050
71812519$S$HW-76323 AND $S$HW-78180 , AND FOUR OF THE                 NSA060
71812519ISOTOPES ARE IDENTIFIED AS HAVING ADVANTAGES                  NSA070
71812519GREAT ENOUGH TO JUSTIFY FURTHER CONSIDERATION WITH OR         NSA080
71812519WITHOUT ISOTOPIC PURIFICATION . $M$THESE PROMISING            NSA090
71812519ISOTOPES ARE $M$CS$E1$134 , $M$HO$E1$166 ,                    NSA100
71812519$M$TM$E1$170 , AND $M$TL$E1$204 , AND OF THESE ,              NSA110
71812519$M$TL$E1$204 IS BELIEVED TO BE THE MOST PROMISING ,           NSA120
71812519IF AN EFFECTIVE SEPARATION PROCESS FOR THIS                   NSA130
71812519ISOTOPE CAN BE ACHIEVED . ( $S$D.L.C. ) .                     NSA140/

==RIF==1812521
=$CAT=$001
41812521$S$HOT $S$LABORATORY $S$DIVISION .                            NSA010/
=$CAT=$002
71812521$M$EFFORTS TO PREPARE ULTRAHIGH SPECIFIC                      NSA020
71812521ACTIVITY $M$MG$E1$28 BY RECOIL TECHNIQUES ESTABLISHED         NSA030
71812521THAT RECOIL DOES OCCUR , THE RANGE OF THE MOST                NSA040
71812521ENERGETIC $M$MG$E1$28 ATOM BEING 0.2 TO 1.2 MG/CM$E1$2 ,      NSA050
71812521EFFECTING AN ENRICHMENT OF ABOUT 100 TO 200 .                 NSA060
71812521$M$THE EXCITATION FUNCTIONS FOR THE                           NSA070
71812521$M$MG$E1$26$N$(T$TC$C$N$P)$M$MG$E1$28 AND                     NSA080
71812521$M$MG$E1$25$N$(T$TC$C$TG$A$N$)$M$NA$E1$24 REACTIONS WERE      NSA090
71812521MEASURED . $M$THE $M$AR$E1$40$N$($TG$A$TC$C$N$2P)$M$AR$E1$42  NSA100
```

```
71812521REACTION WAS OBSERVED , AND EVIDENCE FOR THE                 NSA110
71812521$M$AR$E1$40$N$(T$TC$C$N$P)$M$AR$E1$42 REACTION IS          NSA120
71812521BEING SOUGHT . $M$THE THICK TARGET YIELD OF THE              NSA130
71812521$M$CR$E1$50$N$($TG$A$TC$C$N$2N)$M$FE$E1$52 REACTION         NSA140
71812521WAS MEASURED AS 8.17 $TG$M$S$C/$TG$M$N$AH AS EFFORTS         NSA150
71812521TO PREPARE THIS PRODUCT WERE CONTINUED . $M$THE              NSA160
71812521CAUSE OF OCCASIONAL LOW YIELDS FROM $M$SR$E1$87M             NSA170
71812521GENERATORS IS BEING INVESTIGATED . $M$COMPARISON OF          NSA180
71812521REACTIONS INDUCED BY TRITONS IN TARGETS OF MEDIUM            NSA190
71812521$S$Z IN ACCELERATORS ON THE ONE HAND AND IN                  NSA200
71812521REACTORS ON THE OTHER IS IN PROGRESS . $S$A STUDY            NSA210
71812521OF THE POSSIBILITY OF ELECTRONICALLY SIMPLIFYING GAMMA       NSA220
71812521SPECTRA PRIOR TO PULSE-HEIGHT ANALYSIS IS BEING CONTINUED .  NSA230
71812521$M$AS$E1$72 CAN BE MILKED IN 44( YIELDS FROM                 NSA240
71812521DIAMINONAPHTHALENE-$M$SE$E1$72 , BUT , THUS FAR ,             NSA250
71812521ONLY ACCOMPANIED BY 10( $M$SE$E1$72 CONTAMINATION .           NSA260
71812521$M$AN EXTENDED TABLE OF NUCLIDIC MASSES WAS                  NSA270
71812521CONSTRUCTED IN A WAY SUCH THAT THE STANDARD                  NSA280
71812521DEVIATION BETWEEN CALCULATED AND MEASURED MASSES             NSA290
71812521WAS ONLY 0.35 $M$MEV FOR 763 CASES . $S$A                    NSA300
71812521STUDY OF THE BEHAVIOR OF $M$TE AND $M$MO ON                  NSA310
71812521ALUMINA INDICATED THAT THE VALENCE STATE OF $M$TE MAY BE A   NSA320
71812521FACTOR WHICH WILL HAVE TO BE CONTROLLED MORE CLOSELY IN      NSA330
71812521CONNECTION WITH THE PREPARATION OF $S$I$E1$132 AND           NSA340
71812521$M$TC$E1$99M GENERATORS . $M$IN A STUDY OF THE               NSA350
71812521EFFECT OF CHELATION ON THE ATOMIC SPIN-ORBITAL               NSA360
71812521COUPLING IN A HEAVY ATOM SUCH AS AN ACTINIDE                 NSA370
71812521ELEMENT , SINGLET-TRIPLET TRANSITIONS IN ANTHRACENE          NSA380
71812521COULD NOT BE INDUCED BY MIXING SINGLET AND TRIPLET           NSA390
71812521STATES OF THE AROMATIC WITH THE CHARGE TRANSFER              NSA400
71812521STATES OF THE COMPLEX BETWEEN ANDTHRACENE AND                NSA410
71812521FERRIC ACETYLACETONATE . $S$A FREE RADICAL OF                NSA420
71812521THE COMPLEX BETWEEN $M$SE AND 1$TC$C$N$2-DIAMINOBENZENE      NSA430
71812521WAS PREPARED . $M$POLAROGRAPHIC AND CHRONOPOTENTIOMETRIC      NSA440
71812521INVESTIGATIONS INDICATED THAT TWO DIFFERENT                  NSA450
71812521$S$U(V) SPECIES ARE FORMED , AS A RESULT OF                  NSA460
71812521ELECTROREDUCTION OF $S$U(VI) AND ELECTROOXIDATION OF         NSA470
71812521$S$U(IV) , RESPECTIVELY . $M$DOUBLE LAYER                    NSA480
71812521AND ADSORPTION EFFECTS WERE ENCOUNTERED IN A                 NSA490
71812521POLAROGRAPHIC STUDY OF THE BISMUTH-$S$EDTA COMPLEX .          NSA500
71812521$M$COMPUTER ANALYSIS OF CURRENT REVERSAL                     NSA510
71812521CHRONOPOTENTIOMETRY TOGETHER WITH AN IMPROVED                NSA520
71812521LABORATORY TECHNIQUE INCREASED BY A FACTOR OF 10 THE         NSA530
71812521RANGE OF APPLICATION IN DETERMINING SECOND-ORDER             NSA540
71812521RATE CONSTANTS . $M$PYROCATECHOL VIOLET WAS FOUND TO BE A    NSA550
71812521SUITABLE INDICATOR FOR VOLUMETRIC DETERMINATION OF           NSA560
71812521$M$TH . $M$THE PROBLEM OF   LOW   YIELDS OF                  NSA570
71812521$S$I$E1$132 WAS SOLVED WHEN THE DIFFICULTY WAS               NSA580
71812521TRACED TO SLOW PRECIPITATION OF $M$TE$E1$132 FROM            NSA590
71812521A BASIC SOLUTION ON STANDING . $M$CONDITIONS WERE            NSA600
71812521DETERMINED BY WHICH $S$I$E1$131$S$/I$E1$132 RATIO IN          NSA610
71812521RADIOIODINE MILKED FROM $S$I$E1$132 GENERATORS               NSA620
71812521CAN BE REDUCED TO AS LOW AS 0.014( . $M$ADSORPTION            NSA630
71812521OF $M$TC$E1$99M ONTO SMALL PARTICLES OF INERT MATERIAL IS     NSA640
71812521BEING STUDIED AS A POSSIBLY SUPERIOR REPLACEMENT FOR         NSA650
71812521COLLOIDAL GOLD AND $S$I$E1$131$N$-LABELED $M$ROSE            NSA660
71812521$M$BENGAL FOR CERTAIN MEDICAL APPLICATIONS . $M$THE          NSA670
```

71812521SI$E1$132 PRODUCTION SCHEDULE WAS CHANGED IN A WAY NSA680
71812521THAT OBVIATES DIFFICULTIES CAUSED BY DELAYS IN NSA690
71812521RESTARTING THE REACTOR . SA NEUTRON GENERATOR WAS NSA700
71812521ACQUIRED AND PRELIMINARY STEPS TO INSTALL IT HAVE NSA710
71812521BEGUN . MPLASTIC GLOVE BOXES WERE REINFORCED NSA720
71812521AS A RESULT OF TESTS CARRIED OUT TO DETERMINE NSA730
71812521EFFECTS OF INCREASED INTERNAL PRESSURE . SD NSA740
71812521WASTE INVENTORY REACHED A LOW OF 23TCCN000 GAL . NSA750
71812521(AUTH) . NSA760/

==FIF==1812522
=$CAT=$001
41812522MNONCORROSIVE MPACKING ON AN MISOTOPIC NSA010
41812522MEXCHANGE MCOLUMN FOR MENRICHMENT OF SN-15 . NSA020/
=$CAT=$002
71812522MTHE PROPERTIES OF FOAM GLASS GRANULES (0.5-4 MM NSA030
71812522DIAMETER) AS A CORROSION RESISTANT PACKING OF AN NSA040
71812522ISOTOPE EXCHANGE COLUMN FOR ENRICHMENT OF SN$E1$15 NSA050
71812522WERE INVESTIGATED . MTHE GEOMETRICAL SURFACE OF THE NSA060
71812522PACKING , THE FLOODING CAPACITY OF THE COLUMN , NSA070
71812522THE LOST HEAD , AND THE TRAPPING AND HEIGHT OF A NSA080
71812522THEORETICALLY EQUIVALENT DISK WERE DETERMINED . NSA090
71812522(TR-AUTH) . NSA100/

==FIF==1812523
=$CAT=$001
41812523SSIMPLE SAND SRAPID SPRODUCTION SOF NSA010
41812523SCARRIER-FREE SSODIUM-24 . NSA020/
=$CAT=$002
71812523SA SIMPLE AND RAPID METHOD IS DESCRIBED FOR THE NSA030
71812523PRODUCTION OF CARRIER-FREE SODIUM-24 FOLLOWING THE NSA040
71812523IRRADIATION OF MAGNESIUM WITH FAST NEUTRONS ACCORDING TO THE NSA050
71812523NUCLEAR EQUATION MMG$E1$24N(NTCCNP)MNA$E1$24 . NSA060
71812523MBASED ON THE IRRADIATION OF MMGSO OR NSA070
71812523MMGS(OH)$I1$2 IN THE PRESENCE OF WATER FOLLOWED BY NSA080
71812523EXTRACTION OF MNA$E1$24 ALSO WITH WATER AND WITH A NSA090
71812523FINAL PURIFICATION STAGE , INVOLVING ION EXCHANGE , NSA100
71812523THE DESCRIBED METHOD SHOULD PROVE VERY SUITABLE NSA110
71812523FOR WORK AT THE CURIE LEVEL . MDETAILED DATA ARE NSA120
71812523GIVEN FOR THE PRE-PURIFICATION OF THE COMMERCIAL NSA130
71812523MAGNESIUM TARGET MATERIAL AND FOR ITS PREPARATION AS A NSA140
71812523SLURRY SO AS TO TAKE ADVANTAGE OF RECOIL PRODUCTS PRODUCED NSA150
71812523DURING THE IRRADIATION . MCURVES AND DATA ARE PRESENTED NSA160
71812523SHOWING THE YIELD DEPENDENCE ON THE WATER CONTENT OF THE NSA170
71812523MAGNESIUM TARGET AND OF THE EFFICIENCY OF WATER NSA180
71812523AS AN EXTRACTION MEDIUM . MIN CONCLUSION , NSA190
71812523THE FINAL PURITY OF MNA$E1$24 SEPARATED USING NSA200
71812523THE DESCRIBED METHOD IS DISCUSSED . (AUTH) . NSA210/

```
NUMBER OF CYCLE PROCESSED IN INPUT                       1
NUMBER OF CURRENT WORDS PROCESSED IN INPUT            1213
NUMBER OF NON-WORD ITEMS PROCESSED IN INPUT            262
NUMBER OF DIFFERENT WORDS PROCESSED IN INPUT           440
NUMBER OF ERRORS ENCOUNTERED DURING ANALYSIS             0
EODAD CONDITION ON INPUT TEXT - NO RESTART
```

LIST OF DIFFERENT WORD-ITEMS SELECTED BY TEXT ANALYIS PROGRAM

```
.
[
$E1$
$I1$
$TC$
$TG$
$M$
$N$
$S$
]

,

=$CAT=$
==RIF==

ACETYLACETONATE
ACCELERATORS
APPLICATIONS
ACCOMPANIED
ANDTHRACENE
APPLICATION
ADSORPTION
ADVANTAGES
ANTHRACENE
ACCORDING
ADVANTAGE
ACHIEVED
ACQUIRED
ACTINIDE
ACTIVITY
ANALYSIS
AROMATIC
ALUMINA
AQUEOUS
ATOMIC
ABOUT
ADDED
ALSO
ATOM
AUTH
AND
ANY
ARE
AN
AS
AT
A
BEHAVIOR
BELIEVED
BISMUTH-
BETWEEN
BROMIDE
BROMINE
BENGAL
BASED
BASIC
BEGUN
BEING
BOXES
```

BUT
BE
BY
CHRONOPOTENTIOMETRIC
CHRONOPOTENTIOMETRY
CONSIDERATION
CONTAMINATION
CARRIER-FREE
CONSTRUCTED
CALCULATED
CLASSIFIED
COMMERCIAL
COMPARISON
CONCLUSION
CONDITIONS
CONNECTION
CONTACTING
CONTROLLED
CANDIDATE
CHELATION
COLLOIDAL
COMPRISES
CONSTANTS
CONTINUED
CORROSION
CAPACITY
COMPUTER
COUPLING
CAESIUM
CARRIED
CERTAIN
CHANGED
CLOSELY
COMPLEX
CONTENT
CURRENT
CAUSED
CESIUM
CHARGE
COLUMN
CURVES
CASES
CAUSE
COULD
CURIE
CAN
DIAMINONAPHTHALENE-
DETERMINATION
DIFFICULTIES
DETERMINING
DEPENDENCE
DETERMINED
DIFFICULTY
DESCRIBED
DETERMINE
DEVIATION
DIFFERENT
DISCUSSED
DETAILED
DIAMETER
DIVISION

DENSITY
DILUENT
D.L.C.
DELAYS
DOUBLE
DURING
DATA
DISK
DOES
D
ELECTROOXIDATION
ELECTROREDUCTION
ELECTRONICALLY
ENCOUNTERED
ESTABLISHED
EFFICIENCY
ENRICHMENT
EQUIVALENT
ESPECIALLY
EXCITATION
EXTRACTION
EFFECTING
EFFECTIVE
ENERGETIC
EXTRACTED
EQUATION
EVIDENCE
EXCHANGE
EXTENDED
EFFECTS
EFFORTS
ELEMENT
EFFECT
ENOUGH
EDTA
FOLLOWING
FUNCTIONS
FLOODING
FOLLOWED
FISSION
FURTHER
FACTOR
FERRIC
FORMED
FINAL
FORMS
FOUND
FAST
FLOW
FOAM
FOUR
FREE
FROM
FAR
FOR
GEOMETRICAL
GENERATORS
GENERATOR
GRANULES
GAMMA
GIVEN

GLASS
GLOVE
GREAT
GOLD
GAL
HEAT-SOURCE
HW-76323
HW-73130
HAVING
HEIGHT
HIGHLY
HEAVY
HAND
HAVE
HEAD
HOT
INVESTIGATIONS
IMPROVEMENTS
INVESTIGATED
IRRADIATION
IDENTIFIED
INCREASED
INDICATED
INDICATOR
INVENTORY
INVOLVING
IMPROVED
INCLUDED
INCREASE
INTERNAL
ISOTOPES
ISOTOPIC
INDUCED
INSTALL
ISOTOPE
INERT
ION
ITS
IF
IN
IS
IT
JUSTIFY
LABORATORY
LAYER
LEVEL
LESS
LOST
LOW
MAGNESIUM
MATERIAL
MEASURED
MEDICAL
MASSES
MEDIUM
METALS
METHOD
MILKED
MIXING
MG/CM
MADE

MORE
MOST
MAY
MEV
MM
MO
NITROBENZENE
NONCORROSIVE
NEUTRONS
NUCLIDIC
NEUTRON
NUCLEAR
N-15
NOT
OCCASIONAL
OBSERVED
OBVIATES
OCCUR
OTHER
ONLY
ONTO
ONE
OUT
OF
ON
OR
PRE-PURIFICATION
POLAROGRAPHIC
PRECIPITATION
POLYBROMIDES
PULSE-HEIGHT
PURIFICATION
PYROCATECHOL
POSSIBILITY
PRELIMINARY
PREPARATION
PREFERABLY
PRODUCTION
PROPERTIES
PARTICLES
PRESENTED
PROMISING
POSSIBLY
PREPARED
PRESENCE
PRESSURE
PRODUCED
PRODUCTS
PROGRESS
PACKING
PLASTIC
PREPARE
PROBLEM
PROCESS
PRODUCT
PURITY
PRIOR
PROVE
RE-EXAMINATION
RADIOISOTOPES
RESPECTIVELY

RADIOIODINE
REPLACEMENT
REINFORCED
RESTARTING
REACTIONS
REQUIRING
RESISTANT
REACTION
REACTORS
RECOVERY
RELATING
REMOVING
REVERSAL
RUBIDIUM
RADICAL
REACHED
REACTOR
REDUCED
RECOIL
RESULT
RANGE
RAPID
RATIO
RATE
ROSE
SINGLET-TRIPLET
SECOND-ORDER
SPIN-ORBITAL
SIMPLIFYING
SEPARATING
SEPARATION
SELECTION
SEPARATED
SODIUM-24
STRIPPING
SCHEDULE
SOLUTION
SPECIFIC
STANDARD
STANDING
SUITABLE
SUPERIOR
SHOWING
SINGLET
SOLVENT
SPECIES
SPECTRA
STUDIED
SURFACE
SHOULD
SIMPLE
SLURRY
SOLVED
SOUGHT
STATES
SHEET
SMALL
STAGE
STATE
STEAM
STEPS

STUDY
SALT
SLOW
SUCH
SE
SO
THEORETICALLY
TRANSITIONS
TECHNIQUES
TECHNIQUE
TOGETHER
TRANSFER
TRAPPING
TARGETS
TR-AUTH
TRIPLET
TRITONS
TARGET
TRACED
TABLE
TESTS
THESE
THICK
TAKE
THAN
THAT
THIS
THUS
THE
TWO
TE
TH
TO
ULTRAHIGH
U(IV)
U(VI)
USING
U(V)
VOLUMETRIC
VALENCE
VIOLET
VERY
WATER-IMMISCIBLE
WHEREBY
WITHOUT
WASTE
WATER
WHICH
WERE
WHEN
WILL
WITH
WORK
WAS
WAY
YIELDS
YIELD
Z
0.014(
0.5-4
0.35

0.2
1.2
10(
100
1(
10
1
200
23
2
44(
763
8.17

NOTICE TO THE READER

All scientific and technical reports published by the Commission of the European Communities are announced in the monthly periodical **"euro-abstracts"**. For subscription (1 year : B.Fr 1 025,—) or free specimen copies please write to .

Office for Official Publications
of the European Communities
Boîte postale 1003
Luxembourg
(Grand-Duchy of Luxembourg)

To disseminate knowledge is to disseminate prosperity — I mean general prosperity and not individual riches — and with prosperity disappears the greater part of the evil which is our heritage from darker times.

Alfred Nobel

SALES OFFICES

The Office for Official Publications sells all documents published by the Commission of the European Communities at the addresses listed below, at the price given on cover. When ordering, specify clearly the exact reference and the title of the document.

UNITED KINGDOM

H.M. Stationery Office
P.O. Box 569
London S.E. 1 — Tel. 01-928 69 77, ext. 365

BELGIUM

Moniteur belge — Belgisch Staatsblad
Rue de Louvain 40-42 — Leuvenseweg 40-42
1000 Bruxelles — 1000 Brussel — Tel. 512 00 26
CCP 50-80 — Postgiro 50-80

Agency :
Librairie européenne — Europese Boekhandel
Rue de la Loi 244 — Wetstraat 244
1040 Bruxelles — 1040 Brussel

DENMARK

J.H. Schultz — Boghandel
Møntergade 19
DK 1116 København K — Tel. 14 11 95

FRANCE

Service de vente en France des publications des Communautés européennes — Journal officiel
26, rue Desaix — 75 732 Paris - Cédex 15e
Tel. (1) 306 51 00 — CCP Paris 23-96

GERMANY (FR)

Verlag Bundesanzeiger
5 Köln 1 — Postfach 108 006
Tel. (0221) 21 03 48
Telex: Anzeiger Bonn 08 882 595
Postscheckkonto 834 00 Köln

GRAND DUCHY OF LUXEMBOURG

Office for Official Publications of the European Communities
Boîte postale 1003 — Luxembourg
Tel. 4 79 41 — CCP 191-90
Compte courant bancaire: BIL 8-109/6003/200

IRELAND

Stationery Office — The Controller
Beggar's Bush
Dublin 4 — Tel. 6 54 01

ITALY

Libreria dello Stato
Piazza G. Verdi 10
00198 Roma — Tel. (6) 85 08
CCP 1/2640

NETHERLANDS

Staatsdrukkerij- en uitgeverijbedrijf
Christoffel Plantijnstraat
's-Gravenhage — Tel (070) 81 45 11
Postgiro 42 53 00

UNITED STATES OF AMERICA

European Community Information Service
2100 M Street, N.W.
Suite 707
Washington, D.C. 20 037 — Tel. 296 51 31

SWITZERLAND

Librairie Payot
6, rue Grenus
1211 Genève — Tel. 31 89 50
CCP 12-236 Genève

SWEDEN

Librairie C.E. Fritze
2 Fredsgatan
Stockholm 16
Post Giro 193, Bank Giro 73/4015

SPAIN

Libreria Mundi-Prensa
Castelló 37
Madrid 1 — Tel. 275 51 31

OTHER COUNTRIES

Office for Official Publications of the European Communities
Boîte postale 1003 — Luxembourg
Tel. 4 79 41 — CCP 191-90
Compte courant bancaire: BIL 8-109/6003/300

OFFICE FOR OFFICIAL PUBLICATIONS OF THE EUROPEAN COMMUNITIES
Boîte postale 1003 — Luxembourg
6583