EOL report / Leon Lukaszewicz and Jurg Nievergelt. -
Lukaszewicz, Leon.
Urbana, Ill. : Dept. of Computer Science, University of Illinois, 1967 [i.e. 1968]

https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993

Creative Commons Attribution-NonCommercial-ShareAlike
http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

This work is protected by copyright law (which includes certain exceptions to the rights of the copyright holder that users may make, such as fair use where applicable under U.S. law), but made available under a Creative Commons Attribution-NonCommercial-ShareAlike license. You must attribute this work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work). This work may be copied, distributed, displayed, and performed - and derivative works based upon it - but for non-commercial purposes only (if you are unsure where a use is non-commercial, contact the rights holder for clarification). If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one. Please check the terms of the specific Creative Commons license as indicated at the item level. For details, see the full license deed at http://creativecommons.org/licenses/by-nc-sa/3.0/.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

LIBRARY
OF THE
UNIVERSITY
OF ILLINOIS

510.84
Il6r
no.237-242
cop.2

MATHEMATICS

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

The person charging this material is responsible for its return on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

University of Illinois Library

MAR 18 1971
MAR 22 REC'D
DEC 7 1972
NOV 9 REC'D

L161—O-1096

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by INTERNET ARCHIVE

Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by the Internet Archive
in 2013

http://archive.org/details/eolreport241luka

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

510.84
Il6r
no.241

Math

Report No. 241

EOL REPORT

by

Leon Lukaszewicz

and

Jurg Nievergelt

UNIVERSITY OF ILLINOIS
AUG 15 1968
LIBRARY

September 1, 1967

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Report No. 241

EOL REPORT*

by

Leon Lukaszewicz

and

Jurg Nievergelt

September 1, 1967

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*This work was supported in part by the National Science Foundation under Grant No. GP-4636.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

ACKNOWLEDGEMENT

EOL has been implemented on an IBM 7094 by Freda S. Fischer, M. Irwin-Zarecki, J. R. Sidlo, D. S. Edgar, and R. Sanders. Their active participation in the project is gratefully appreciated.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

TABLE OF CONTENTS

	Page
ACKNOWLEDGEMENT	iii
1. INTRODUCTION	1
1.1. Purpose and history of EOL	1
1.2. The EOL Computer	2
1.2.1. Input and Output	2
1.2.2. Stacks	4
1.2.3. Instruction Address Stack	5
1.2.4. Logical Register H	5
1.2.5. Files	6
1.3. General structure of EOL programs	7
1.4. Formal description of syntax	8
1.5. Reference language and its representation	9
1.6. Notation for the values of variables	10
2. BASIC SYMBOLS	11
2.1. Characters	11
2.1.1. Letters	11
2.1.2. Digits	11
2.1.3. Delimiters	11
2.1.4. Precedence of characters	12
2.2. Markers	12
3. VARIABLES AND THEIR VALUES	13
3.1. Inputs	13
3.2. Outputs	13
3.3. Stacks	14
3.3.1. Words	14
3.3.1.1. Precedence of words	14
3.3.2. Numbers	14
3.3.3. Address	15
3.3.3.1. Instruction address	15
3.3.3.2. Pointer position	15
3.3.4. Examples of stack values	15
3.3.5. Instruction address stack	15
3.4. Files	16
3.5. Logical values	16
3.6. Initial values	16
4. GENERAL FORMAT OF INSTRUCTIONS	17
4.1. Operators	17
4.2. Arguments	17
4.2.1. Literals	17
4.2.2. Integers	18

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by INTERNET ARCHIVE

Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

TABLE OF CONTENTS (Continued)

Page

4.2.3. Indexed mode symbols. 18
4.2.3.1. Intermediate list 18
4.2.3.2. Mode symbols. 19
4.2.3.2.1. Mode of reading . . 19
4.2.3.2.2. Mode of adding. . . 20
4.2.3.2.3. Mode of testing . . 20
4.2.3.3. Indexes 21
4.2.3.4. Examples. 21
4.2.4. Tests 22
4.2.4.1. Number of steps 22
4.2.4.2. Constituent test. 22
4.2.4.2.1. Class 22
4.2.4.2.2. Word test operator. 23
4.2.4.2.3. Number test operator. 24
4.2.4.2.4. Constituent compared. 24
4.2.4.2.5. Address test. . . . 25
4.2.4.3. Record test 25
4.2.4.4. Examples. 25
4.2.5. Labels. 26
4.3. Termination of reading a list 26
4.3.1. Reading an input or a stack 26
4.3.2. Reading a file. 27
4.4. Instruction conventions 28
4.4.1. Missing second argument 28
4.4.2. Testing with an empty stack 28
4.4.3. Undefined result. 29

5. INSTRUCTIONS. 30

5.1. Stack instructions. 30
5.1.1. Move. 30
5.1.2. Exchange. 31
5.1.3. Set into stack. 32
5.1.4. Clear stack 32
5.1.5. Test stack. 33
5.1.6. Arithmetic instructions 34
5.1.7. Convert to word 35
5.1.8. Convert to number 36
5.1.9. Split 37
5.1.10. Compress. 38
5.1.11. Count 38
5.2. Input instructions. 39
5.2.1. Read. 39
5.2.2. Clear input 40
5.2.3. Test input. 41

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by INTERNET ARCHIVE

Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

TABLE OF CONTENTS (Continued)

Page

5.3. Output instructions 42
5.3.1. Write 42
5.3.2. Set output. 42
5.3.3. Output format instructions. 43
5.4. File instructions 44
5.4.1. Put 44
5.4.2. Get 45
5.4.3. Save. 46
5.4.4. Restore 46
5.4.5. Set into file 47
5.4.6. Clear file. 47
5.4.7. Test file 48
5.4.8. Shift 49
5.4.9. Shift and count 50
5.4.10. Copy. 51
5.5. Jump instructions 51
5.5.1. Simple jump 52
5.5.2. Conditional jump. 52
5.5.3. Switch jump 53
5.5.4. Return. 54
5.5.5. Stop. 55
5.6. Table instructions. 55
5.6.1. Search. 55
5.6.2. Search and count. 56
5.6.3. Extract 57
5.7. Macro instructions. 58
5.8. Code. 59

6. DECLARATIONS. 60

6.1. Switches. 60
6.1.1. Name switch 60
6.1.2. Index switch. 60
6.2. Tables. 61
6.2.1. Word table. 61
6.2.2. Number table. 61
6.3. Entry declaration 61
6.4. External declaration. 61
6.5. Index declaration 62

7. PROGRAMS, PROCEDURES, MACROS. 63

7.1. Programs. 63
7.2. Procedure and statement 63
7.2.1. Comments. 64
7.3. Macro definition. 64

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by INTERNET ARCHIVE

Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

TABLE OF CONTENTS (Continued)

		Page
7.4.	Block structure and the scope of declarations . .	66
7.4.1.	Procedures: external and internal. . . .	66
7.4.2.	Declaration of labels	67
7.4.3.	Scope of a declaration of a label	67
7.4.4.	Scope of an index declaration	68
7.4.5.	Use of labels and formal indexes.	68
7.4.6.	Example	69
7.5.	Flow of control in a program.	69
BIBLIOGRAPHY.		71
APPENDIX I.	SUMMARY OF EOL SYNTAX	72
APPENDIX II.	INDEX OF TERMS.	78

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by INTERNET ARCHIVE

Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

1. INTRODUCTION

1.1. Purpose and history of EOL

EOL is a low-level language for manipulating strings of characters. The following considerations guided the design of the EOL language.

(a) It should be relatively easy to learn.

(b) It should be easily implemented on most computers.

(c) EOL programs should run efficiently.

(d) It should be well suited for two of the most important applications of string manipulation languages, namely:

(i) compiler writing

(ii) symbolic and algebraic manipulation

Concerning (a), EOL can be considered to be the assembly language of the fictitious EOL computer (to be described in Section 1.2.), which has some fifty basic instructions and a very simple general structure.

Concerning (b), the EOL system consists of a compiler (see [8]), written in EOL, and an interpreter (see [9]), documented in a language X1 (see [10]) which is close to the assembly language of many modern computers.

Concerning (c), the data structure of EOL is such that on a binary, word oriented computer (for which EOL is mainly intended), operations on strings are done word by word, not character by character. A second feature which enhances the efficiency of EOL is the possibility to call procedures written in machine language from EOL programs.

Concerning (d), reference [7] gives some examples of EOL programs which are discussed in detail. These examples are also intended to help the reader learn EOL. Another example of an EOL application is the EOL compiler itself (see [8]).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Many concepts used in EOL were taken from other symbol manipulation languages, in particular IPL-V [1] and COMIT [2]. The basic features of EOL first appeared in [5]. Two previous versions, EOL-1 and EOL-2 have been developed and implemented at the Institute for Mathematical Machines in Warsaw, Poland, beginning in 1965. On this basis, an improved EOL-3 has been developed and implemented on an IBM 7094 computer at the University of Illinois.

1.2. The EOL Computer

The EOL language can be described as the programming language of the hypothetical EOL computer, shown in Figure 1. The EOL computer is intended to be simulated on a conventional computer and is therefore a pure software system. Binary fixed length word computers are best suited for simulation of the EOL computer.

1.2.1. Input and Output

The EOL computer is equipped with a number of inputs, identified by the variables

$$I0, I1, \ldots, I\alpha$$

and with outputs identified by variables

$$Q0, Q1, \ldots, Q\beta$$

In practice, inputs and outputs correspond to such devices as card readers, line printers, or magnetic tapes. Therefore, the numbers α and β are usually small integers. Input and output data is represented in the form of strings of arbitrary characters, like letters, digits, arithmetic operators, etc. (see sections 3.1. and 3.2.).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

EOL COMPUTER

Figure 1

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

The input data can be read under EOL program control, character by character, and each character read is deleted from the input string. Strings of consecutive characters read from an input string can be assembled into larger units called words, which may consist of an arbitrary number of characters. Input strings can be tested under program control for occurrence of an arbitrarily specified character pattern. Due to this, no limitations are imposed upon the form of the input data so that the user may choose the format most convenient for his purpose.

Similarly, the output strings are constructed under program control and presented to an output by adding new characters at the end of the indicated output string.

1.2.2. Stacks

Internal processing in the EOL computer is done mostly on a number of stacks, identifed by the variables

$$S0, S1, \ldots, S\gamma$$

The number γ depends on the particular implementation, and typically is equal to 31.

Stacks are linear lists of units called constituents, which are strings of characters preceded by one of the marks $^{-}$ or ^{o} or $\wedge$. These marks are not part of the EOL character set, but serve to indicate the structure of a list and the types of its constituents.

A constituent may be of the type word, number, or address.

A word consists of a string of arbitrary characters, preceded by the mark $^{-}$.

A number is a freely chosen string of digits, possibly with an algebraic sign and preceded by the mark ^{o}. Numbers are usually identified with the fixed point numbers available on the particular computer on which EOL is implemented. They permit one to perform the usual arithmetic operations on integers.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

An address is represented by any string of characters preceded by the mark ^. Addresses are usually identified with addresses available on the particular computer on which EOL is implemented. They are used to identify

(a) a record in a file

(b) a position of the pointer in a file

(c) an instruction in an EOL program

The syntax of stack values is described in section 3.3.

In the EOL language there are many instructions which permit flexible processing of stacks. For example,

(a) Moving a determined number of constituents from the beginning of one stack to the beginning or to the end of another stack, the original order of constituents being kept or reversed.

(b) Compressing several words into one word or splitting one word into separate words of one character each.

(c) Testing whether the initial or the final constituent of a stack is equal to a given word or number.

In an EOL implementation, stacks are held in the main memory as lists composed of pairs of computer words. Storage allocation for stacks including retrieval of abandoned pairs for reuse proceeds dynamically and automatically, using a "garbage collection" technique.

1.2.3. Instruction Address Stack

A special instruction address stack IAS is used to hold return addresses of subroutine calls.

1.2.4. Logical Register H

A one bit register H is used to hold the outcome (success or failure) of tests. H can be tested by conditional transfer instructions. The two possible values of the variable H are denoted by "plus" and "minus". A successful test leaves the value of H

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

unchanged, an unsuccessful one sets it to "minus". Thus the effect of several consecutive test instructions is that H will be "minus" if at least one of the tests failed (logical OR). All conditional transfer instructions automatically reset H to "plus", whether or not the condition is satisfied.

1.2.5. Files

In the EOL computer mass storage is provided by files, identified by the variables.

P0, P1, ..., P8

Usually, the files P0 and P1 are held in core storage, and the others correspond to such back-up storage devices as discs, drums, or magnetic tapes.

Each of the files represents a linear list composed of any number of records.

A normal record consists of a string of constituents, preceded by the mark $\vee$. The entire content of a stack or part of it may be stored as a normal record in a file.

A special record consists of a mark $\vee$ followed by the symbol σ. It is used to denote the boundary between different sections of a file.

In every file there is exactly one pointer $\uparrow$ which indicates some place or some record in the file. Reading from or writing into a file always concerns the record which follows directly after the pointer.

The syntax of file values is described in section 3.4.

In the EOL language there are many instructions which permit one to process files, for example:

(a) Shift the pointer forward until it is positioned immediately before the first-record that satisifies a specified condition.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

(b) Store the address of the record which is placed directly after the pointer as the initial constitutent of an indicated stack.

(c) Place the pointer of a file directly in front of the record whose address is the initial constituent of an indicated stack.

The last two operations permit one to manipulate addresses of records by means of instructions designed to process stacks. This facility is very important in many applications where an effective search for information depends on the data structure in a file.

Files are designed to store a large amount of information. For example, in program translation they may serve for storing tables of identifiers or processed programs.

1.3. General structure of EOL programs

An EOL program consists of a sequence of macro definitions followed by a sequence of external procedures. Macro definitions can be included in external procedures at compile time. A macro definition alone or an external procedure along with all macro definitions which are to be included in this external procedure are units which can be compiled separately (independent of any other macro definition or external procedure).

Both a macro definition and a procedure consists of a heading (which includes, among other things, the symbols MACRO or PROC), a procedure body, and a termination symbol END.

A procedure body is a sequence of statements, and a statement in turn may be:

(a) An instruction, which usually causes the execution of a simple action such as a change of the value of a variable. However, a macro instruction, which causes the insertion at compile time of a copy (possibly modified by the replacement of actual for formal parameters) of a macro

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

definition in place of this macro instruction, may be arbitrarily complex.

(b) A procedure. The fact that a procedure may contain statements which are again procedures gives rise to the nested block structure of EOL.

(c) Declarations, used to declare switches, tables of constants (numbers or literals), formal indexes, or labels with a special function (external labels or entry points into a procedure).

(d) Comments, which have no effect on the execution of programs.

Execution of an EOL program begins with the first instruction of the procedure declared to be MAIN (see 7.2.), and proceeds to the instruction written next, unless the present instruction (a jump) or the statement written next (a non executable statement) explicitly state otherwise.

In the EOL language programs as well as input and output data have the form of strings of characters. In particular any EOL program may be the input to or the result of another program written in this language.

1.4. Formal description of syntax

This report uses the Backus notation (which was designed for the description of ALGOL [3]) to describe the syntax of EOL programs and data. This notation has been extended with the following conventions:

{a|b...|z}	denotes "precisely one of the symbols a,b... z"
[a|b...|z]	denotes "none or one of the symbols a,b... z"
{a}..	denotes "an arbitrary non-empty sequence of symbols a"

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

[a]..	denotes "an arbitrary, possibly empty sequence of symbols a"

Examples:

{A|B}	may denote A
{A|B}..	may denote ABA
[A|B]..	may denote an empty sequence

1.5. Reference language and its representation

The EOL language as defined in this report will be called the reference language. Both the program and the data are defined in this language as sequences of basic symbols satisfying some specified syntax rules. Typographical features such as a space or line feed are not part of the reference language.

In order to allow compatibility with available equipment and clarity for a user, the EOL reference language admits various representations. In such a representation each symbol of the reference language has a character or typographical feature associated with it.

In the examples of programs contained in this report, the following representation of the reference language is used.

(1) The symbol ␣ is replaced by an arbitrary non-empty string of blanks.

(2) Each <delimiter> can be surrounded from both sides by any number of blanks.

(3) The symbol λ denotes carriage return and line feed.

(4) If the last nonblank character in a line is the character "-", then the next line is treated as a continuation of the current line and not as a new line.

(5) The remaining characters of the representation are the same as in the reference language.

The above convention is not used for auxiliary notations for the values of variables as described in 1.6., where the symbol ␣ preserves its

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

form, while blanks and line feeds have no meaning.

1.6. Notation for the values of variables

In the examples given in this report the following notation is used to illustrate the value of a variable:

<variable name> : <variable value> [⊣ |⊢]

The symbol ⊣ means that the elements (e.g. characters, constituents, records) occur in the sequence <variable value> in the order in which they are written from left to right. The symbol ⊢ means that these elements occur in the sequence in the order in which they are written from right to left. If none of these symbols occur the effect is the same as if the sequence were terminated by ⊣ . This latter symbol is often used to emphasize the end of a sequence.

Examples:

I1 : ABCD
I3 : ⊣
S4 : ¯A1¯=¯3 ⊣
S4 : ¯3¯=¯A1 ⊢

The latter two notations are equivalent. It should be stressed here that this type of notation is of auxiliary significance only and has no influence on program execution or the data structure.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

2. BASIC SYMBOLS

The EOL reference language is built up of the following basic symbols

<basic symbol>::={<character>|<marker>}

2.1. Characters

<character>::={<letter>|<digit>|<delimiter>}

2.1.1. Letters

<letter>::={A|B...|Z}

The above set can be extended arbitrarily with other letters, e.g. lower case letters.

2.1.2. Digits

<digit>::={0|1|2 .....|9}

2.1.3. Delimiters

<delimiter>::={␣|,|:|'|*|λ|γ|/}

The above list can be extended arbitrarily with any other delimiters. In the reference language the character λ serves the purpose of separating consecutive statements in an EOL program, while γ is assigned no function. In particular implementations these delimiters may denote typographical features, e.g. λ may denote "carriage return line feed" for output on a printer and "end of card" for input from a card reader. γ may denote "start new page" on output or the "escape character" used in several character codes.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

2.1.4. Precedence of characters

The order of precedence is defined for the EOL character set by the following list:

$\sqcup$
A,B,..... Z
0,1,..... 9
λ, γ

We say that a character which on the above list occurs earlier precedes any character which occurs later and that the later character follows the earlier one.*

2.2. Markers

$$\langle\text{marker}\rangle ::= \{ {}^{-} \mid {}^{\circ} \mid {}^{\vee} \mid \sigma \mid {}^{\wedge} \mid \uparrow \}$$

Markers, except the pointer $\uparrow$, serve to define the structure of stacks and files as described later. The pointer serves to distinguish a certain record in a file.

---

*The precedence of the characters which do not occur in this list is undefined. It may be defined in a particular implementation.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

3. VARIABLES AND THEIR VALUES

EOL programs operate on a limited number of variables selected from the following set:

inputs	I0, I1,...Iα
outputs	Q0, Q1,...Qβ
stacks	S0, S1,...Sγ
files	P0, P1,...Pδ
logical variable	H
instruction address stack	IAS

α, β, γ, δ denote same fixed poitive integers which depend on the particular implementation. For instance, $\alpha = \beta = 1$, $\gamma = 31$, $\delta = 3$.

3.1. Inputs

<input value>::=<character string>

<character string>::=[<character>]..

Examples:

I1 : JOHN ␣ SMITH

I3 : A = B * (C + D) ␣ λ ⊣

3.2. Outputs

<output value>::=<character string>

Example:

Q2 : X = 3.14159; ␣ END

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

3.3. Stacks

<stack value>::=[<constituent>]..

<constituent>::={<word>|<number>|<address>}

3.3.1. Words

<word>::=¯{<character>}..

Examples:

¯ABC ¯X=A+B ␣ λ ; ¯38

3.3.1.1. Precedence of words

We say that the word X precedes the word Y if in the process of matching them character by character from left to right, in the first pair of different characters the character of the word X precedes the corresponding character of the word Y (see 2.1.4.).

We assume that absence of a character (at the end of a word) precedes any character.

We say that the word X follows the word Y if the word Y precedes the word X.

3.3.2. Numbers

<number>::=°<integer>

<integer>::=[+|-]{<digit>}..

Examples:

°36 °-48 °0006

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

3.3.3. Address

<address>::={<instruction address>|<pointer position>}

3.3.3.1. Instruction address

<instruction address>::= ^{<character>}..

An instruction address is a symbol which denotes the location of an instruction in a program. Instruction addresses are used automatically by procedure calls and returns (see section 5.5.). They may be operated upon under program control by exchanging the values of the instruction address stack IAS and of another stack (see 5.1.2.).

3.3.3.2. Pointer position

<pointer position>::= ^{<character>}..

A pointer position is a symbol which denotes the beginning or the end of a file or the boundary between two records in a file. It may be used to save the current position of a file pointer and to restore the pointer to that place later.

3.3.4. Examples of stack values

S1 : ¯X1¯=¯A¯*¯(¯B¯+¯GAMMA¯)
S7 : ¯Z¯+¯Y¯=¯X ⊢
S16 : °13^P08¯λ

3.3.5. Instruction address stack

The instruction address stack IAS is used mainly for storing addresses of instructions of the type CALL. These addresses can be used later by the RETURN instruction (see 5.5.).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Example:

$$IAS : \hat{}38\hat{}A63\hat{}18 \dashv$$

3.4. Files

$$\langle\text{file value}\rangle ::= [\langle\text{record}\rangle]..\uparrow[\langle\text{record}\rangle]..$$

$$\langle\text{record}\rangle ::= \{{}^{v}\langle\text{stack value}\rangle \mid {}^{v}\sigma\}$$

The symbol σ makes possible division of the file into sections. Therefore, each file can consist of an arbitrary number of sequentially ordered sections.

Examples:

$$P1 : \uparrow$$
$$P3 : {}^{v}{}^{-}A{}^{-}BC^{o}38\uparrow$$
$$P6 : {}^{v}a{}^{v}b{}^{v}c{}^{v}\sigma\uparrow{}^{v}d{}^{v}e{}^{v}f$$

where a,b,... denote stack values.

3.5. Logical value

$$\langle\text{logical value}\rangle ::= \{+ \mid -\}$$

Examples:

H : + H : -

3.6. Initial values

By convention at the beginning of each program run the variable H has the value "+" while all stacks (including IAS) are empty. The initial values for inputs, outputs and files must be defined for each particular run.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

4. GENERAL FORMAT OF INSTRUCTIONS

The form of an EOL instruction is either

<operator> ␣ [<argument>[,<argument>[,<argument>]]]

or

<operator> ␣ <argument>,,<argument>

The latter is a special case of a three-argument instruction where the second argument need not be written because it is assigned a standard value (see 4.4.1.).

4.1. Operators

An operator is always a string of capital letters, e.g.

MOVE, SET, SEARCH

The operator determines an instruction or a class of instructions which are differentiated more exactly by the form of their arguments.

4.2. Arguments

<argument>::={<literal>|<integer>|<indexed mode symbol>|
<test>|<label>}

4.2.1. Literals

<literal>::={'<any string of characters with the restriction that quotation marks can occur only in adjacent pairs. Such a pair inside the literal denotes one quotation mark>'|{L|G}*}.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

In the above definition

L* denotes the character λ

G* denotes the character γ

Examples of literals

'A'	denotes A
'␣␣␣'	denotes three blanks
'A''A'	denotes A'A
''''''	denotes ''
L*	denotes λ

4.2.2. Integers

See section 3.3.2.

4.2.3. Indexed mode symbols

<indexed mode symbol>::=<mode symbol>×<index>

4.2.3.1. Intermediate list

Many EOL instructions serve to transform some list of elements. Execution of these instructions can be divided into three steps:

(1) The successive elements of the source list indicated by the first argument of an instruction are read to form an intermediate list. The possible modes of reading a source list are described in 4.2.3.2.1.

(2) The intermediate list is transformed in a way determined by the instruction operator.

(3) The result is added to the destination list indicated by the second argument of the instruction. The possible modes of adding the intermediate list to a destination list are described in 4.2.3.2.2.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Some instructions cause reading of at most one element of the source list, others may cause reading of an arbitrary number of elements. The moment when reading the source list is to be terminated is defined by a test indicated by the last argument of the instruction. For further discussion on the intermediate list see section 4.3.

4.2.3.2. Mode symbols

<mode symbol>::={A|B|Y|Z|T|C|D|I|Q}

The <mode symbol> specifies the mode in which certain steps of the execution of an instruction are performed. In some instructions the mode symbol indicates also the type of variable to be operated upon.

4.2.3.2.1. Mode of reading

The mode of reading a source list to form an intermediate list is defined by the mode symbols as follows:

A	read and delete successive constituents from the beginning of a stack S.
B	read and copy successive constitutents from the beginning of a stack S without deleting them.
Z	read and copy the last constituent from a stack S without deleting it.
I	read and delete successive characters from the beginning of input I.
C	read and delete successive records following the pointer ↑ in a file P.
D	read and copy successive records following the pointer ↑ in a file P without deleting them. The pointer is then placed after the last record which has been read.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

4.2.3.2.2. Mode of adding

After an intermediate list has been created and perhaps transformed then it is added to the destination list in a way defined by the mode symbol as follows:

A	reverse the order of the constituents in the intermediate list, then attach this modified list at the beginning of stack S.
B	attach the intermediate list of constitutents at the beginning of a stack S.
Y	reverse the order of the constituents in the intermediate list, then attach this modified list at the end of a stack S.
Z	attach the intermediate list of constituents at the end of a stack S.
C	reverse the order of the records in the intermediate list, then insert this modified list in the file P directly after the pointer ↑.
D	insert the intermediate list of records in a file P directly after the pointer ↑, then place the pointer ↑ directly after the last record of the inserted list.
Q	attach the intermediate list of characters at the end of output Q.

4.2.3.2.3. Mode of testing

Indexed symbols are also used to indicate a constituent which is tested against another constituent defined by the instruction.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

In this case the meaning of the indexed symbol is as follows:

B	test using the first constituent of a stack S.
Z	test using the last constituent of a stack S.
T	test using consecutively all the constituents in a stack S. The test is said to be fulfilled if it is fulfilled in relation to at least one constituent of S.

4.2.3.3. Indexes

<index>::={<actual index>|'<formal index>}

<actual index>::={1|2|....Ω}

where Ω is equal to the maximum of α, β, γ, δ.

<formal index>::=<letter>[<letter>|<digit>]..

To facilitate the description of instructions, the following symbols are introduced to denote indexes:

<n>::=<index>

<m>::=<index>

<k>::=<index>

4.2.3.4. Examples

Indexed mode symbols with actual indexes:

A1, B8, A7, P1

Indexed mode symbols with formal indexes:

A'X, B'C2, P'AB

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

4.2.4. Tests

<test>::={<number of steps>|<constituent test>|<record test>}

In most instructions the <test> argument defines the moment when the process of forming an intermediate list should be terminated. This argument is always placed as the last argument in an instruction.

4.2.4.1. Number of steps

<number of steps>::=*{<integer>|{B|Z}<k>}

The <number of steps> is determined either by <integer> or the first or the last constituent of stack S<k>. The absolute value of this number indicates the number of elements to be read from a source list to form an intermediate list.

4.2.4.2. Constituent test

<constituent test>::{<word test>|<number test>|<address test>}

<word test>::={<class>|<word test operator>
{<literal>|<constituent compared>}}

<number test>::=<number test operator>
{<integer>|<constituent compared>}

<address test>::=<constituent compared>

A <constituent test> defines a condition which may or may not be fulfilled by a constituent.

4.2.4.2.1. Class

<class>::={<class letter>}..

<class letter>::={L|D|B|R}

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

In the sequence which forms the <class> symbol each <class letter> may appear only once. The meaning of the <class letter> is as follows:

L denotes letters
D denotes digits
B denotes the character ␣ ("blank")
R denotes all remaining characters

The classes specified by the above symbols are mutually exclusive and they exhaust all the characters used in EOL.

A <class> symbol denotes a set of characters. A character belongs to this set if it belongs to one of the classes specified by the <class letters> used in this symbol.

Examples:

LD	denotes all letters and digits
RB	denotes all characters except letters and digits
LRB	denotes all characters except digits

When testing a word against a <class> symbol the first character only in the tested word is considered. For instance, the test of a word with the class symbol DL is fulfilled if the first character of this word is a digit or a letter. Otherwise, the test fails.

4.2.4.2.2. Word test operator

<word test operator>::=[{PR|PE|FL|FE}/]

The word test operator specifies what relation is to be tested between the word indicated by an instruction and the word indicated by <literal> or <constituent compared>.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

The indicated options of <word test operator> have the following meaning:

Operator	Relation
empty	equal
PR/	precedes
PE/	precedes or equal
FL/	follows
FE/	follows or equal

4.2.4.2.3. Number test operator

<number test operator>::=[{GT|GE|LT|LE}/]

The <number test operator> specifies what relation is to be tested between the number indicated by an instruction and the number indicated by <literal> or <constituent compared>.

The indicated options of <number test operator> have the following meaning:

Operator	Relation
empty	equal
GT/	greater than
GE/	greater than or equal
LT/	less than
LE/	less than or equal

4.2.4.2.4. Constituent compared

<constituent compared>::={B|Z|T}<k>

The <constituent compared> specifies a constituent or a list of constituents which are to be tested. The meaning of the letters B, Z, and T is explained in (4.2.3.2.3.).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

4.2.4.2.5. Address test

The <address test> specifies a constituent or a list of constituents (assumed to be addresses) which are to be tested. The address test has no operator and, hence, tests for equality.

4.2.4.3. Record test

<record test>::={<constituent test>|S*|R*}

(1) A record test of the type <constituent test> is fulfilled if the first constituent in the record which is located directly after the pointer in the indicated file satisfies the test.

(2) A record test of the type S* is fulfilled if a special record σ is located directly after the pointer in the indicated file.

(3) A record test of the type R* is fulfilled if the special record σ is located just before the pointer in the indicated file.

The file in which the record is to be tested is determined by the instruction which contains the <record test>.

4.2.4.4. Examples

The test symbols written on the left test a constituent (which is defined by the instruction which contains this constituent test) for the condition specified on the right.

B3	equal to the first constituent in S3. Both constituents are assumed to be of the same type--words, numbers or addresses.
LT/T2	less than at least one of the constituents of S2. Both constituents are assumed to be numbers.
PE/Z5	precedes or is equal to the last constituent in S5. Both constituents are assumed to be words.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

If the two constituents tested are not of the same type, the result of the test is undefined (see 4.4.3.).

4.2.5. Labels

<label>::=<a non-empty sequence of characters different from ␣ |,|:|'>

Labels are used in programs to denote instructions, switches, procedures and macro-definitions.

Examples:

```
ALFA
*
X1+2
```

4.3. Termination of reading a list

The first step in executing many instructions consists of forming an intermediate list by reading the consecutive elements of a source list indicated by the first argument of an instruction (see 4.2.3.1.). This action is terminated when either all the elements of the list (up to some definite point) have been read or when the conditions defined by the test argument are fulfilled.

In some cases, when no elements are read from the source list, the intermediate list remains empty. For example, this is the case when the source list is initially empty or when the first element of this list fulfills the test. In all these cases execution of the instruction is terminated without performing any other of its steps, possibly with the exception that H is set to minus.

4.3.1. Reading an input or a stack

This section includes the cases of reading the consecutive constituents in a stack and of reading the consecutive characters either in an input string or in a constituent of type word. The termination of the reading is determined by the following rules.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

(1) If the <test> argument is absent, then all the elements of the indicated list are read. In particular, if this list is initially empty then execution of the instruction is immediately terminated without changing any of the variables.

(2) If the test argument is <number of steps> then the reading is terminated when the number of elements which have been read is equal to the number defined by the test argument.

(3) If the test argument is <constituent test> then the reading is terminated with the last constituent which does not fulfill the test. Therefore, the first constituent not read is the one which fulfills the test.

If the first element of a list fulfills the test, no element is read and execution of the instruction is terminated without changing any of the variables.

(4) If the test argument is present and all the constituents of a stack have been read before the indicated test is fulfilled then the reading is terminated and H is set to minus. If the source list is initially empty then execution of the instruction is terminated and H is set to minus.

4.3.2. Reading a file

Termination of the reading of consecutive records in a file is governed by the following rules.

(1) If the <test> argument is absent, then the reading is terminated when all the records between the pointer ↑ and the end of the file have been read. In particular, when the pointer ↑ is initially at the end of the file then execution of the instruction is immediately terminated without changing any variables.

(2) If the test argument is <number of steps> then the reading is terminated when the number of the records which have been read is equal to the number deinfed by that argument, unless conditions described in paragraph (4) arise.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

(3) If the test argument is <constituent test> then the reading of a file is terminated with the last record which does not fulfill the test. If the record just after the pointer fulfills the test, no record is read and execution of the instruction is terminated. Other conditions are described in paragraph 4.

(4) If the test argument is of the type <number of steps> or <constituent test> and all the records between the pointer ↑ and the nearest record σ or the end of the file have been read before the indicated test is fulfilled, then the reading is terminated and H is set to minus.

(5) If the test argument is equal to S* then the reading of a file is terminated just before a special record σ is read or when an end of file is met.

(6) If the test argument is equal to R* then the reading of a file is terminated just after a special record σ has been read. If the record σ is not met, then all the records between the pointer ↑ and the end of file are read and H is set to minus.

4.4. Instruction conventions

The following conventions are valid for all the instructions described in section 5. even if they are not specifically mentioned.

4.4.1. Missing second argument

Some stack instructions have the format

<operator> ␣ <argument>,,<argument>

in which case the second argument is said to be missing. By convention, each such instruction has the same effect as one in which the second argument is present and equal to B<n>, where <n> is the index that occurs in the first argument.

4.4.2. Testing with an empty stack

If the last argument of an instruction is of the type <test> and it refers to a stack which is empty, then the only effect of this

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

instruction is to set the value of H to minus.

4.4.3. Undefined result

In the description of many instructions certain implicitly or explicitly mentioned assumptions are made. For instance:

(1) In arithmetic instructions it is assumed that the quantities defined by two argument of the instruction are numbers.

(2) In all tests it is assumed that the two quantities to be compared are of the same type, e.g. they are both words, both numbers or both record addresses.

(3) It is assumed that any instruction which uses a label lies within the scope of exactly one declaration of this label.

In the case when such assumptions are not fulfilled, the result of the execution of the instruction is undefined.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

5. INSTRUCTIONS

<instruction>::={<stack instruction>|<input instruction>|
<output instruction>|<file instruction>|
<jump instruction>|<table instruction>|
<macro instruction>|<code>}

5.1. Stack instructions

<stack instruction>::={<move>|<exchange>|<set into stack>|
<clear stack>|<test stack>|
<arithmetic instruction>|<convert to word>|
<convert to number>|<split>|<compress>|<count>}

5.1.1. Move

<move>::=MOVE ␣ {A|B|Z}<n>, {A|B|Y|Z}<m>
[,{<number of steps>|<constituent test}]

Read successive constituents from S<n> and add them to S<m>.

The first argument specifies the mode of reading constituents from stack S<n> (see 4.2.3.2.1.).

The second argument specifies the mode of adding to stack S<m> the constituents which have been read (see 4.2.3.2.2.).

The third argument (or its absence) specifies the moment when reading of constituents from S<n> is to be terminated (see 4.3.1.).

Examples:

Let us assume that

S1 : ¯XA¯1¯=¯18°3¯;

S3 : ¯ALFA¯ ␣

S8 : ¯;¯=

H : +

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Execution of the MOVE instruction on the left causes the results indicated on the right.

MOVE A1, A3, *2	S1 :	¯=¯18°3¯;
	S3 :	¯1¯XA¯ALFA¯ ␣
MOVE B1, B3, '='	S3 :	¯XA¯1¯ALFA¯ ␣
MOVE B1, A3, R	S3 :	¯1¯XA¯ALFA¯ ␣
MOVE B1, Z3, T8	S3 :	¯ALFA¯ ␣ ¯XA¯1
MOVE A1, Y3, Z8	S1 :	¯=¯18°3
	S3 :	¯ALFA¯ ␣ ¯1¯XA
MOVE A3, B3, B	no changes	
MOVE A3, B3, D	H :	-
MOVE B3, B3	S3 :	¯ALFA¯ ␣ ¯ALFA¯ ␣

5.1.2. Exchange

<exchange>::=EXCHANGE ␣ {B<n>|IAS},{B<m>|IAS}

Exchange the contents of the stacks indicated by the first and second argument.

Example:

Let

S1	:	¯X1¯Q
S8	:	¯X2°3
IAS	:	^L1^L2

Execution of the following instructions causes:

EXCHANGE B1, B8	S1 :	¯X2°3
	S2 :	¯X1¯Q
EXCHANGE IAS, B1	S1 :	^L1^L2
	IAS :	¯X1¯Q

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

EXCHANGE IAS, IAS no changes

Exchanging the instruction address stack IAS allows the programmer to operate on the return addresses of his subroutine calls.

5.1.3. Set into stack

<set into stack>::=SET ␣ {B|Z}<n>,{<integer>|<literal>}

Add to S<n> the constituent whose value is indicated by the second argument. In the case of <integer> the constituent added is of type number. In the case of <literal> it is of type word.

The first argument specifies the mode of adding the constituent to S<n> (see 4.2.3.2.2.).

Examples:

Let S5 : $^{-}X^{o}3$

Execution of the following instructions causes:

SET B5, 7	S5 : $^{o}7^{-}X^{o}3$
SET Z5, 'ALFA'	S5 : $^{-}X^{o}3^{-}$ALFA

5.1.4. Clear stack

<clear stack>::=CLEAR ␣ A<n>[,{<number of steps>|<constituent test>}]

Read and delete consecutive constituents from the beginning of stack S<n>.

The second argument (or its absence) specifies the moment when reading and deleting of constituents from S<n> is to be terminated (see 4.3.1.).

Example:

Let

S3 : $^{-}XA^{-}=^{-}18$

S4 : $^{o}1^{o}3^{o}5$

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Execution of the following instructions causes:

CLEAR A3, *2	S3 : $^{-}18$
CLEAR A4, GT/4	S4 : $^{o}5$

5.1.5. Test stack

<test stack>::=TEST ␣ {A|B|Z}<n>[,<constituent test>]

If the second argument is present and S<n> is not empty then read one constituent from S<n> (see 4.2.3.2.1.) and test if it fulfills the relationship specified by the second argument (see 4.2.4.2.). If the test fails, then assign the value minus to H.

If the second argument is present and S<n> is empty, then assign the value minus to H.

If the second argument is absent and S<n> is not empty, then assign the value minus to H. Furthermore, if the first argument is equal to A<n>, delete the first constituent from S<n>.

If the second argument is absent and S<n> is empty nothing changes. Thus a test stack with a missing second argument tests whether a stack is empty.

Examples:

Let

S1 : $^{-}X1^{o}8$ ⊣
S5 : ⊣
H : +

Execution of the following instructions causes:

TEST A1, D	S3 : $^{o}8$ ⊣
	H : -
TEST B1, LB	H : no change
TEST Z1, GT/8	H : -

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

TEST A5	H	: no change
TEST Z5	H	: no change

5.1.6. Arithmetic instructions

<arithmetic instructions>::={ADD|SUB|MULT|DIV} ␣ {A|B|Z}<n>,
[{B|Z}<m>] , {<integer>|{B|Z}<k>}

Perform the arithmetic operation indicated by the operator on the operands specified by the first and the third argument. Put the result on stack S<m>.

If the second argument in the instruction is omitted then the result is added at the beginning of S<n>.

The first argument specifies the mode of reading S<n> to obtain the first operand (see 4.2.3.2.1.).

The third argument specifies either an integer or the mode of reading S<k> to obtain the second operand.

The second argument indicates the mode of adding the result to S<m> (see 4.2.3.2.2.).

The meaning of the operators is as follows:

ADD	add
SUB	subtract (the operand specified by the third argument from the operand specified by the first argument)
MULT	multiply
DIV	divide (the operand specified by the first argument by the operand specified by the third argument)

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

ADD, SUB and MULT yield only one constituent as result. DIV yields two constituents as a result. The first is the remainder of the division, which has the same sign as the dividend. The second is the integral part of the quotient. If the divisor is zero the division is not performed and H is set to minus. If the stack indicated by either the first or the third argument is empty then H is set to minus.

Examples:

Let

S1 : $^{o}9^{-}A^{o}0$

S3 : $^{o}-11^{-}3$

Execution of the following instructions causes:

ADD B1, Z1, B3	S1 : $^{o}9^{-}A^{o}0^{o-}2$
MULT A1,,2	S1 : $^{o}18^{-}A^{o}0$
DIV A3,, Z1	S3 : no change H : -
DIV A3, B3, B1	S1 : no change S3 : $^{o}-2^{o}-1^{-}3$
SUB B1, B3, Z3	result undefined (the constituent indicated by the third argument is not of type number)

5.1.7. Convert to word

<convert to word>::=WORD ␣ {A|B|Z}<n>[,{B|Z}<m>]

If S<n> is empty, set H to minus. Otherwise, read one constituent (which is supposed to be of type number) from S<n> (see 4.2.3.2.1.), convert it to a constituent of type word which expresses this number in decimal notation and add the result to S<m> (see 4.3.2.2.2.). The word obtained is an integer (see 4.2.2.) which

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

has no leading zeros (with the exception of zero which results in a single digit zero) and no plus sign if it is positive.

If the second argument in the instruction is omitted then the result is added at the beginning of S<n>.

Examples:

Let

S3 : $^{o}6^{o}7$

S5 : ^{o}-11^{-}A

Execution of the following instructions causes:

WORD B3, Z3	S3 : $^{o}6^{o}7^{-}6$
WORD A5	S5 : $^{-}$-11^{-}A
WORD Z5	result undefined

5.1.8. Convert to number

<convert to number>::=NUMBER ␣ {A|B|Z}<n>[,{B|Z}<m>]

If S<n> is empty, set H to minus. Otherwise, read one constituent (which is supposed to be a word of the form <integer> from S<n> (see 4.2.3.2.1.), convert it to a constituent of type number whose value is given by the integer in S<n>, and add the result to stack S<m> (see 4.3.2.2.2.).

If the second argument in the instruction is omitted then the constituent obtained is added at the beginning of S<n>.

Examples:

Let

S3 : $^{-}$+11^{-}A

S5 : $^{-}$-07^{-}B

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Execution of the following instructions causes:

NUMBER B3, B5 S5 : $^{o}11^{-}-07^{-}B$

NUMBER A5 S5 : $^{o}-7^{-}B$

5.1.9. Split

<split>::=SPLIT ␣ {A|B|Z}<n>[,{A|B|Y|Z}<m>|
,[{A|B|Y|Z}<m>],{<number of steps>|<word test>}]

If stack S<n> is empty, set H to minus. Otherwise, read from S<n> one constituent (which is assumed to be a word) and form a sequence of constituents each of which consists of a single character in such a way that the first character of the constituent read forms the first constituent in the new sequence, the second character forms the second constituent, etc., until a termination criterion (specified by the third argument) is satisfied. Then add this sequence of constituents to S<m>. The first argument in the instruction specifies the mode of reading from S<n> (see 4.2.3.2.1.).

The second argument specifies the mode of adding to S<m> the sequence of constituents formed from the constituent which was split (see 4.2.3.2.2.). If the second argument is omitted this sequence is added to the beginning of S<n> (with order preserved). Thus a missing second argument is equivalent to a second argument B<n>.

The third argument specifies the moment when splitting is terminated. The word read from S<n> is treated as a sequence of characters and the rules of reading of section 4.3.1. are applied. If the third argument is absent, then the whole word is split.

Examples:

Let

S1 : $^{-}AB=3^{-}CC$

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Execution of the following instructions causes:

SPLIT A1	S1 :	¯A¯B¯=¯3¯CC
SPLIT B1, Z1, *1	SL :	¯AB=3¯CC¯A
SPLIT A1,,'='	S1 :	¯A¯B¯CC

5.1.10. Compress

<compress>::=COMPRESS ␣ {A|B}<n>[,{B|Z}<m>|,[{B Z}<m>],
{<number of steps>|<word test>}]

If S<n> is empty, set H to minus. Otherwise, read successive constituents (all of which are assumed to be of type word) from S<n>, concatenate these to form one new constituent of type word, and add the result to S<m>.

The first argument specifies the mode of reading S<n> (see 4.2.3.2.1.).

The second argument specifies the mode of adding the result to S<m>. The absence of this argument is equivalent to the case when it is equal to B<n>.

The third argument (or its absence) specifies the moment when reading of constituents from S<n> is to be terminated. If it is missing, then all constituents of S<n> are read.

5.1.11. Count

<count>::=COUNT ␣ {A|B}<n>[,{B|Z}<m>|,[{B|Z}<m>],<constituent test>]

If S<n> is empty set H to minus. Otherwise, read successive constituents from S<n> and generate a constituent of the type number whose value is equal to the number of constituents just read. Then add this constituent to S<m>.

The first argument specifies the mode of reading constituents from S<n> (see 4.2.3.2.1.).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

The second argument specifies the mode of adding the resulting constituent to S<m> (see 4.2.3.2.2.). The absence of this argument is equivalent to the case when it is equal to B<n>.

The third constituent (or its absence) specifies the moment when reading of constituents from S<n> is to be terminated.

Examples:

Let

S3 : ¯AY¯BX¯CZ

S4 : °3°4

The instructions given below cause:

COUNT A3, B4, 'BX'	S3 :	¯BX¯CZ
	S4 :	°1°3°4
COUNT A4,,5	S4 :	°2
	H :	-
COUNT B4	S4 :	°2°3°4

5.2. Input instructions

<input instruction>::={<read>|<clear input>|<test input>}

5.2.1. Read

<read>::=READ␣I<n>,{B|Z}<m>[,{<number of steps>|<word test}]

Read and delete successive characters from input I<n>, concatenate these characters to form one word, and add this word as one constituent to S<m>. The order of characters in the word being formed is the same as the order in which they are read.

The second argument specifies the mode of adding to S<m> the word which has been formed (see 4.2.3.2.2.).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

The third argument (or its absence) specifies the moment when reading of input characters is to be terminated (see 4.3.1.).

Examples:

Let

```
I1  :  X1 = 28; λ␣⊣
S3  :  ¯PQ ⊣
H   :  +
```

Execution of the instructions given below causes:

```
READ I1, B3, *5          I1  :  ; λ␣⊣
                         S3  :  ¯X1 = 28 ¯PQ

READ I1, Z3, R           I1  :  = 28; λ␣⊣
                         S3  :  ¯PQ¯X1

READ I1, B3 '.'          I1  :  ⊣
                         S3  :  ¯X1 = 28; λ␣¯PQ
                         H   :  -
```

5.2.2. Clear input

<clear input>::=CLEAR ␣ I<n>[,{<number of steps>|<word test>}]

Read and delete successive characters from I<n>.

The second argument (or its absence) specifies the moment when reading of input characters is to be terminated (see 4.3.1.).

Examples:

Let

```
I3  :  λ␣␣SKIP␣⊣
H   :  +
```


Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Execution of the instructions given below causes:

CLEAR I3, 'SKIP'	I3 : SKIP ␣ ⊣
CLEAR I3, D	I3 : ⊣
	H : -

5.2.3. Test input

<test input>::=TEST ␣ I<n>[,<word test>]

If the second argument is present and I<n> is not empty then read without deletion one character from I<n> and regard it as a word. If this word does not satisfy the relation defined by the second argument set H to minus.

If the second argument is present and I<n> is empty, set H to minus.

If the second argument is absent and I<n> is not empty set H to minus.

If the second argument is absent and I<n> is empty then nothing changes. Thus an input test with a missing second argument tests whether an input is empty.

Examples:

Let

I1 : X1 = 14 ⊣
I5 : ⊣
H : +

Execution of the following instructions causes:

TEST I1, LR	H : no change
TEST I1	H : -
TEST I5, LR	H : -
TEST I5	H : no change

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

5.3. Output instructions

<output instruction>::={<write>|<set output>|<output format instruction>}

5.3.1. Write

<write>::=WRITE ␣ {A|B|Z}<n>, Q<m>[,{<number of steps>|<word test>}]

Read successive constituents (all of which are assumed to be of type word) from S<n> and add them one after another to the end of output Q<m>, in the same order in which they occurred in S<n>.

The third argument (or its absence) specifies the moment when reading of constituents from S<n> is to be terminated (see 4.3.1).

Examples:

Let

S7 : ¯X1¯=¯13¯ ␣ ¯λ ⊣

Q2 : RESULT ␣ ⊣

Execution of the following instructions causes:

WRITE A7, Q2	S7 : ⊣
	Q2 : RESULT ␣ X1 = 13 ␣ λ
WRITE B7, Q2, R	Q2 : RESULT ␣ X1
WRITE A7, Q2, *9	S7 : ⊣
	Q2 : RESULT ␣ X1 = 13 ␣ λ
	H : -

5.3.2. Set output

<set output>::=SET ␣ Q<n>,<literal>

Add to the end of Q<n> the literal indicated by the second argument.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Examples:

Let

Q3 : X = ⊣

Execution of the following instructions causes:

SET Q3, '13'	Q3 : X = 13 ⊣
SET Q3, 'UNDEF'	Q3 : X = UNDEF ⊣

5.3.3. Output format instructions

<output format instruction>::={BLANK|LINE|PAGE} ␣ Q<m>,<number of steps>

Perform the indicated operation as many times as specified by the second argument.

The instruction operators have the following meaning:

BLANK	Put blank characters at the end of Q<m>
LINE	Skip to the beginning of a new line on the output Q<m>
PAGE	Skip to the beginning of a new page on the output Q<m>

(The operators LINE and PAGE are dependent on the particular EOL implementation; e.g., they may be defined only for the printer, not for other output devices.)

If in the hardware representation the characters λ and γ mean skipping to a new line or to a new page, respectively, then the last two operators have the following meaning:

LINE	Put the character λ at the end of Q<m>
PAGE	Put the character γ at the end of Q<m>

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Example:

Let

Q3 : X = 10 ⊣

Execution of the following instructions causes:

BLANK Q3, *4	Q3 : X = 10 ␣␣␣␣ ⊣
LINE Q3, *2	Q3 : X = 10 λλ ⊣
	(provided λ means line feed carriage return)

5.4. File instructions

<file instruction>::={<put>|<get>|<save>|<restore>|
<set into file>|<clear file>|
<test file>|<shift>|<shift and count>|
<copy>}

5.4.1. Put

<put>::=PUT ␣ {A|B|Z}<n>,{C|D}<m>[,{<number of steps>|<constituent test>}]

Read successive constituent from S<n>, form one record of them and place it in the file P<m>.

The first argument specifies the mode of reading S<n> (see 4.2.3.2.1.).

The second argument specifies the mode of adding to P<m> the record consisting of the constituents which have been read (see 4.2.3.2.2.).

The third argument (or its absence) specifies the moment when reading of constituents from S<n> is to be terminated (see 4.3.1.).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Example:

Let

$$P2 \ : \ {}^{\vee}a^{\vee}b\uparrow{}^{\vee}c \dashv$$

$$S3 \ : \ {}^{-}A^{\circ}3^{\circ}8 \dashv$$

Execution of the following instructions causes:

PUT B3, C2, *2	P2 : ${}^{\vee}a^{\vee}b\uparrow{}^{-}A^{\circ}3^{\vee}C \dashv$
PUT A3, D2	P2 : ${}^{\vee}a^{\vee}b^{\vee-}A^{\circ}3^{\circ}8\uparrow{}^{\vee}C$
	S3 : $\dashv$

5.4.2. Get

<get>::=GET ␣ {C|D}<n>,{A|B|Y|Z}<m>
[,{<number of steps>|<constituent test>}]

Read the record which follows directly the pointer in P<n>, then read successive constituents from this record and add them to S<m>.

The first argument specifies the mode of reading the record from P<n> (see 4.2.3.2.1.).

The second argument specifies the mode of adding to S<m> the constituents which have been read (see 4.2.3.2.2.).

The third argument (or its absence) specifies the moment when reading of constituents from the record is to be terminated (see 4.3.2.).

Examples:

Let

$$P2 \ : \ {}^{\vee}a\uparrow{}^{\vee-}A^{-}B^{\circ}3^{\vee}C \dashv$$

$$S3 \ : \ {}^{-}XQ \dashv$$

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Execution of the following instructions causes:

GET C2, B2	P2 :	$^{\vee}a\uparrow \quad {}^{\vee}C \dashv$
	S3 :	${}^{-}A\,{}^{-}B\,{}^{o}3\,{}^{-}XQ \dashv$
GET D2, Y3, *2	P2 :	no change
	S3 :	${}^{-}XQ\,{}^{-}B\,{}^{-}A \dashv$

5.4.3. Save

<save>::=SAVE ␣ P<n>,{B|Z}<m>

Read the current position of the pointer in file P<n> and save it as a constituent of type address on stack S<m>.

The second argument specifies the mode of adding this constituent to S<m> (see 4.2.3.2.2.).

5.4.4. Restore

<restore>::=RESTORE ␣ [{A|B|Z}<n>,]P<m>

If the first argument is missing, place the pointer at the beginning of file P<m>. Otherwise, read one constituent (which is assumed to be a pointer position) from S<n>, and place the pointer into the position indicated by this constituent. If S<n> is empty, set H to minus.

The first argument specifies the mode of reading the constituent from S<n>.

Example:

Let

$$P1 : {}^{\vee}a\,{}^{\vee}b\,{}^{\vee}c\,{}^{\vee}\sigma\uparrow\,{}^{\vee}d$$

$$S3 : {}^{\wedge}C\,{}^{-}X$$

where ${}^{\wedge}C$ is the address of the record ${}^{\vee}c$.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Execution of the following instruction causes:

RESTORE P1	P1	: $\uparrow^{\vee}a^{\vee}b^{\vee}c^{\vee}\sigma^{\vee}d$
RESTORE B3, P1	P1	: $^{\vee}a^{\vee}b\uparrow^{\vee}c^{\vee}\sigma^{\vee}d$

5.4.5. Set into file

<set into file>::=SET ␣ {C|D}<n>,{<integer>|<literal>|S*}

Add to the file P<n> the record specified by the second argument. If the second argument is S*, the special record $^{\vee}\sigma$ is added. If it is <integer> or <literal>, a record composed of one constituent only is added. <Integer> defines a constituent of type number and <literal> a constituent of type word.

The first argument specifies the mode of adding the record to P<n> (see 4.2.3.2.2.).

Example:

Let

P1 : $^{\vee}a\uparrow^{\vee}b^{\vee}c$

Execution of the following instructions causes:

SET C1, 3	P1	: $^{\vee}a\uparrow^{\vee\circ}3^{\vee}b^{\vee}c$
SET D1, 'AB'	P1	: $^{\vee}a^{\vee-}AB\uparrow^{\vee}b^{\vee}c$
SET C1, S*	P1	: $^{\vee}a\uparrow^{\vee}\sigma^{\vee}b^{\vee}c$

5.4.6. Clear file

<clear file>::=CLEAR ␣ P<n>[,{<number of steps>|<record test>}]

Read and delete successive records from P<n>, beginning at the current pointer position.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

The second argument (or its absence) specifies the moment when reading (and deleting) records from P<n> is to be terminated (see 4.3.2.).

Example:

Let

P2 : ˇa↑ˇbˇcˇσˇd

H : +

Execution of the following instructions causes:

CLEAR P2, *2	P2 : ˇa↑ˇσˇd
CLEAR P2, *3	P2 : ˇa↑ˇσˇd
	H : -
CLEAR P2, R*	P2 : ˇa↑ˇd ⊣
CLEAR P2	P2 : ˇa↑ ⊣

5.4.7. Test file

<test file>::=TEST ␣ P<n>[,<record test>]

Test file P<n>, and if it does not satisfy the relationship specified by the second argument (or its absence) set H to minus.

The following cases can arise:

(a) If the second argument is missing, test whether P<n> is empty (if not, set H to minus).

(b) If the second argument is a <constituent test>, then test the first constituent of the record which follows directly the pointer. If this constituent does not satisfy the relationship, set H to minus.

(c) If the second argument is S*, test whether the record which follows directly the pointer is ˇσ. If not, set H to minus.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

(d) If the second argument is R*, then test whether the record which immediately <u>precedes</u> the pointer is ˇσ. If not, set H to minus.

<u>Examples</u>:

Let

Pl : ˇaˇσ↑¯A°3ˇC

Execution of the following instructions causes:

TEST Pl, LD	H : no change
TEST Pl, 'B'	H : -
TEST Pl, R*	H : no change
TEST Pl, S*	H : -
TEST Pl	H : -

5.4.8. <u>Shift</u>

<shift>::=SHIFT ␣ P<n>[,{<number of steps>|<record test>}]

Shift the pointer in file P<n> forward. The moment of stopping the pointer is specified by the second argument (or its absence) in the same way as it is when reading the file (see 4.3.2.). In the same circumstances the variable H is also given the value minus.

<u>Examples</u>:

Let

Pl : ˇa↑ˇbˇcˇdˇσˇe ⊣

where

d : ¯ALFA°3^P ⊣

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

while the first constituents in the records b and c are different from ALFA. Then execution of the following instructions causes:

SHIFT P1, 'ALFA'	P1	: $^{\vee}a^{\vee}b^{\vee}c\uparrow^{\vee}d^{\vee}\sigma^{\vee}e$
SHIFT P1, *10	P1	: $^{\vee}a^{\vee}b^{\vee}c^{\vee}d\uparrow^{\vee}\sigma^{\vee}e$
	H	: -
SHIFT P1	P1	: $^{\vee}a^{\vee}b^{\vee}c^{\vee}d^{\vee}\sigma^{\vee}e\uparrow$
	H	: no change

5.4.9. Shift and count

<shift and count>::=SHIFTC ␣ P<n>,{B|Z}<m>
[,{<number of steps>|<record test>}]

Shift the pointer in file P<n> forward. The moment of stopping the pointer is specified by the third argument (or its absence) in the same way as it is when reading a file (see 4.3.2.). Add one constituent of type number, whose value is equal to the number of records skipped by the file pointer, to S<m> in the mode specified by the second argument (see 4.2.3.2.2.).

Example:

Let

P1 : $^{\vee}a\uparrow^{\vee}b^{\vee}c^{\vee}\sigma^{\vee}d$

S3 : $^{-}A1$

Execution of the following instructions causes:

SHIFTC P1, B3, R*	P1	: $^{\vee}a^{\vee}b^{\vee}c^{\vee}\sigma\uparrow^{\vee}d$
	S3	: $^{o}3^{-}A1$
SHIFTC P1, Z3	P1	: $^{\vee}a^{\vee}b^{\vee}c^{\vee}\sigma^{\vee}d\uparrow$
	S3	: $^{-}A1^{o}4$

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

5.4.10. Copy

<copy>::=COPY ␣ {C|D}<n>,{C|D}<m>[,{<number of steps>|<record test>}]

Read successive records in P<n> and add them to P<m>.

The first argument specifies the mode of reading records from file P<n> (see 4.2.3.2.1.).

The second argument specifies the mode of adding to file P<m> the records which have been read (see 4.2.3.2.2.).

The third argument (or its absence) specifies the moment when the reading of records from P<n> is to be terminated (see 4.3.2).

Examples:

Let

P1 : ˇa↑ˇbˇcˇσˇd ⊣

P2 : ˇp↑ˇqˇr ⊣

Execution of the following instructions causes:

COPY C1, D2, *3	P1 :	ˇa↑ˇσˇd ⊣
	P2 :	ˇpˇbˇc↑ˇqˇr
	H :	-
COPY D1, C2, R*	P1 :	ˇaˇbˇcˇσ↑ˇd
	P2 :	ˇp↑ˇcˇbˇqˇr
COPY C1, C2	P1 :	ˇa↑
	P2 :	ˇp↑ˇdˇσˇcˇbˇqˇr

5.5. Jump instructions

<jump instruction>::={<simple jump>|<conditional jump>|
<switch jump>|<return>|<stop>}

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

5.5.1. Simple jump

<simple jump>::={GOTO|CALL} ␣ <label>

Go to the instruction indicated by <label>. CALL in addition causes the address of the instruction in which this operator occurs to be placed as one constituent of type address at the beginning of the instruction address stack IAS.

5.5.2. Conditional jump

<conditional jump>::={GOMI|GOPL|CAMI|CAPL} ␣ <label>

If the condition indicated by the operator of the instruction is fulfilled then go to the instruction indicated by <label>. In either case reset H to plus.

CAPL and CAMI in addition place the address of the instruction in which these operators occur at the beginning of the instruction address stack IAS.

The conditions indicated by the operators are:

Operators	Condition
GOPL, CAPL	value of H is plus
GOMI, CAMI	value of H is minus

Example:

If the value of H is equal to minus, then the instruction

CAMI ALFA

causes placing the address of this instruction on top of the stack IAS, going to the place of the program denoted by the label ALFA and resetting the value of H to plus.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

5.5.3. Switch jumps

<switch jump>::={GONA | GOIN | CANA | CAIN} ␣ {A|B|Z}<n>,<label>

Read one constituent from S<n>, and use this constituent to select a label from the label list of the switch indicated by the second argument of the <switch jump>. Then go to the instruction indicated by the label which was selected. In case of CANA and CAIN, in addition place the address of this instruction <switch jump> at the beginning of IAS.

The first argument specifies the mode of reading the constituent from S<n> (see 4.2.3.2.1.). If S<n> is empty, set H to minus and proceed.

The operator of the instruction specifies the type of switch which is referred to by the second argument as follows:

GONA, CANA refer to a name switch (see 6.1.1.) and assume that the constituent read from S<n> is a word. The label selected from the label list of the switch is equal to the word read from S<n>.

GOIN, CAIN refer to an index switch (see 6.1.2.) and assume that the constituent read from S<n> is a number. The index of the label selected from the label list of the switch is equal to the absolute value of the number read from S<n>. The index of the first label of the label list is defined to be zero.

The process of selecting a label from the label list of the switch (see 6.1.1. and 6.1.2.) may fail because:

(1) stack S<n> is empty, or

(2) the word read from S<n> does not occur in the label list of the name switch or the number read from S<n> is outside the range of legal indexes of the index switch.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

In both cases the switch jump is not executed, H is set to minus and control proceeds to the instruction following this switch jump.

Examples:

Assume a name switch KEY and an index switch LX are declared as follows:

KEY : NAME FOR, IF, BEGIN, END

LX : INDEX LA, LB, LC

Let the values of stacks S1, S2, S3 and S4 be:

S1 : $^{o}0^{-}A^{o}8$

S2 : –

S3 : $^{-}$BEGIN^{-}REAL^{-}X

S4 : ^{o}-2°4

Then the switch jumps on the left have the same effect as the simple jumps listed on the right below:

GOIN B1, LX	GOTO LA
CANA B3, KEY	CALL BEGIN
CAIN B4, LX	CALL LC

while the switch jumps CAIN Z1, LX and CAIN Z2, LX both set H to minus.

5.5.4. Return

<return>::=RETURN

Read and delete the first constituent (which is assumed to be an instruction address) from IAS, and go to the instruction which

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

follows immediately the one whose address was just read. If the instruction address stack IAS is empty, or if its first constituent is not an instruction address, the outcome of RETURN is undefined.

5.5.5. Stop

<stop>::=STOP

Stop program execution.

5.6. Table instructions

<table instruction>::={<search>|<search and count>|<extract>}

5.6.1. Search

<search>::=SEARCH ␣ {A|B|Z}<n>,<label>

Read one constituent from S<n>. If the value of that constituent is present on the list of the table or name switch indicated by the second argument, leave H unchanged. Otherwise, set H to minus.

The first argument of the instruction specifies the mode of reading a constituent from S<n> (see 4.2.3.2.1.).

The second argument determines the label of the table or name switch.

If the indicated table is of the type number table, then the constituent read from S<n> is assumed to be a number. In the case of a word table or name switch this constituent is assumed to be a word. If these conditions are not satisfied, then the result of the instruction is undefined.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Examples:

Let

SWITCH : NAME A, B, C

WORDTABLE : TABLE 'A', 'B', 'C'

NUMBERTABLE : TABLE 3, 7, 11

S1 : $^{o}7^{o}12$

S2 : $^{-}X^{o}3^{-}C$

Execution of the following instructions causes:

SEARCH A1, NUMBERTABLE	S1	: $^{o}12$
	H	: no change
SEARCH Z1, NUMBERTABLE	H	: -
SEARCH Z2, SWITCH	H	: no change
SEARCH Z2, WORDTABLE	H	: no change
SEARCH B1, WORDTABLE	result undefined	

5.6.2. Search and count

<search and count>::=SEARCHC ␣ {A|B|Z}<n>,[{B|Z}<m>],<label>

Perform the instruction <search> using the first argument and the <label>. If the search is successful then the index of the first element in the list of the table or name switch which is equal to the constituent read from S<n>, is added as one constituent to S<m>.

The second argument specifies the mode of adding the resulting constituent to S<m> (see 4.2.3.2.2.). The absence of this argument is equivalent to the case when it would be equal to B<n>.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Examples:

Let

KEY : TABLE 'IF', 'FOR', 'BEGIN', 'END'

S1 : ¯FOR¯REAL

S3 : ¯INDEX

Execution of the following instructions causes:

SEARCHC B1, Z3, KEY	S3 :	¯INDEX°1
SEARCHC A1,,KEY	S1 :	°1¯REAL
SEARCHC Z1,,KEY	H :	-

5.6.3. Extract

<extract>::=EXTRACT ␣ {A|B|Z}<n>,[{B|Z}<m>],<label>

Read one constituent, which is assumed to be a number, from S<n>, and use the absolute value of this number as an index to extract an element from the list of the table or name switch indicated by the third argument. Then add this element as a constituent to S<m>. The index of the first element in a list is zero.

The first argument specifies the mode of reading the constituent from S<n> (see 4.2.3.2.1.).

The second argument specifies the mode of adding the resulting constituent to S<m> (see 4.2.3.2.2.). The absence of this argument is equivalent to the case when it would be equal to B<n>.

The last argument determines the label of the table or name switch.

The element extracted from a number table is added to S<m> as a constituent of the type number. The element extracted from a word table or name switch is added to S<m> as a constituent of the type word.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

If the absolute value of the number read from S<n> is greater than or equal to the number of elements in the table or name switch, then H is set to minus.

Examples:

Let

PX : TABLE 'AB', 'BB', 'ZB', 'AZ', 'BZ', 'ZZ'

S3 : $^{o}3^{o}12$

S5 : $^{-}$DQ

The execution of the following instructions causes:

EXTRACT B3, B5, PX	S5 :	$^{-}$AZ^{-}DQ
EXTRACT A3,,PX	S3 :	$^{-}$AZ^{o}12
EXTRACT Z3,,PX	H :	-

5.7. Macro instructions

<macro instruction>::=<macro name> ␣ <actual parameter list> λ

<macro name>::=<label>

<actual parameter list>::=[<actual parameter>
[,<actual parameter>]..]

<actual parameter>::={[<macro name continuation>']{<actual index>|
<formal index>}|<label>|<literal>}

A macro instruction calls for the insertion, at compilation time, of the procedure body of the macro definition which has the same macro name and macro name continuations, with all formal parameters of the macro definition replaced by the corresponding actual parameters. See section 7.3. on macro definitions.

The actual parameter list in the macro inclusion statement must be compatible with the formal parameter list of the macro definition. I.e., the number of positions in both lists must be the

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

same, identical macro name continuations must occur in corresponding positions, and parameters of the same type (<index>, <label>, or <literal>) must occur in corresponding positions.

5.8. Code

<code>::=CODE ␣ [<n>,[<m>,]]<label>

Go to the external procedure written in machine language specified by the label which is the last argument of the instruction.

The two indexes <n> and <m> can be used to transmit arguments to the external procedure.

Details of combining EOL and machine language programs are necessarily dependent upon the particular computer installation and, hence, must be defined separately for each implementation.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

6. DECLARATIONS

<declaration>::={<switch>|<table>|<external declaration>|
<entry declaration>|<index declaration>}

6.1. Switches

<switch>::={<name switch>|<index switch>}

6.1.1. Name switch

<name switch>::={<label>:}..NAME ␣ <label list>

<label list>::=<label>[,<label>]..

Name switches are used by means of switch jumps described in section 5.5.3. and by table instructions described in section 5.6.

Example of name switch:

```
ID  :  NAME A, AB, +, **, 3
```

6.1.2. Index switch

<index switch>::={<label>:}..INDEX ␣ <label list>

Index switches are used by means of switch jumps described in section 5.5.3.

Example of index-switch:

```
XI  :  INDEX Q, 41, ++
```


Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

6.2. Tables

<table>::={<word table>|<number table>}

Tables are used by means of table instructions (see 5.6.).

6.2.1. Word table

<word table>::={<label>:}..TABLE ␣ <literal>[,<literal>]..

Example:

WT : TABLE 'KEY', '/', L*, ''''

6.2.2. Number table

<number table>::={<label>:}..TABLE ␣ <integer>[,<integer>]..

Example:

NT : TABLE 3, 8, -6, + 12

6.3. Entry declaration

<entry declaration>::={<label>:}..ENTRY

An entry declaration provides an alternative point of entry into a procedure in addition to the normal entry through the procedure name. See 7.4.3. on the scope of label declarations.

6.4. External declaration

<extern>::={<label>:}..EXTERN[␣ <comment string>]

An external declaration declares each of its labels to be external to the external procedure in which this declaration occurs;

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

i.e., that it is the name or an entry point of another external procedure (see 7.1. on external procedures and 7.4.3. on the scope of declarations).

6.5. Index declaration

<index declaration>::={<formal index>:}..EQU ␣ <actual index>

An index declaration assigns an actual index to each one of the formal indexes (see 7.3.) occuring in the declaration (see 4.2.3.3.) everywhere within the scope of this declaration (see 7.4.4.).

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

7. PROGRAMS, PROCEDURES, MACROS

7.1. Programs

```
<program>::=[<macro-definition>]..
            [<external procedure>]..
<external procedure>::=<procedure>
```

A program consists of a sequence of macro-definitions followed by a sequence of external procedures. An external procedure is a procedure which is not contained within any other procedure. It is intended that in EOL implementations each macro-definition alone and each external procedure along with all the macro-definitions which are included in this external procedure forms a unit which can be compiled separately, independently of any other part of the program.

7.2. Procedure and statement

```
<procedure>::={<procedure name>:}..{MAIN|PROC} [ ␣ <comment string>] λ
            <procedure body>END[ ␣ <comment string>] λ

<procedure name>::=<label>

<procedure body>::=[<statement> λ]..

<statement>::={[<label>:]..<instruction>|<declaration>|
            <procedure>|<comment>}
```

In each program exactly one procedure should be distinguished by the key word MAIN (usually, but not necessarily, an external procedure). Program execution starts with this procedure (see 7.5.).

Notice that the definition of procedure is recursive, hence procedures may be nested. Instructions and declarations are the subject of section 5. and 6., respectively.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

7.2.1. Comments

<comment>::={COMMENT|C} ␣ <comment string>

<comment string>::=<an arbitrary sequence of characters that does not begin with ":" and does not contain λ>

Comments have no effect on program execution. They are used only for explanations.

7.3. Macro definition

<macro definition>::={<macro name>:}..MACRO ␣ <formal parameter list> λ

<procedure body>END[␣ <comment string>] λ

<macro name>::=<label>

<formal parameter list>::=[<formal parameter>[,<formal parameter>]..]

<formal parameter>::={[<macro name continuation>']<formal index>|<formal label>|<formal literal>}

<macro name continuation>::=<letter>[<letter>|<digit>]..

<formal index>::=<letter>[<letter>|<digit>]..

<formal label>::=<label>

<formal literal>::=<literal>

Macro-definitions cause the definition of a program section in which <formal index>, <formal label> or <formal literal> can occur, only those, however, which are in <formal parameter list> of this macro definition. A <macro instruction> (see 5.7.) will insert this section into the program before execution begins. The name of a macro-definition can be extended by <macro-name continuation>. Thus two macro-definitions of the same name can be totally different when their names have different continuations.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Macro name continuations may be used in an analogous role as the mode symbols A, B, Y, Z, etc., play in many instructions, such as MOVE.

Examples:

```
ABC  :  MACRO LA, 'X', S'K
        SET B'K, 'X'
        GOMI LA
        RETURN
        END

QX   :  MACRO SZ'P, L, EX
        CLEAR I'L
        ABC EX, '+', S'P
        END
```

The second macro definition contains a macro instruction (see 5.7.) referring to the first macro-definition ABC, hence at compilation time a copy of the procedure body of the first (inner) macro-definition is inserted in place of the macro instruction occurring in the second (outer) macro-definition.

In such cases of macro instructions that occur in macro-definitions, replacement of actual for formal parameters proceeds by passing parameters from outer to inner macros.

Example:

Let us suppose that macros ABC and QX are defined according to the examples above. Then the macro instructions given below on the left are replaced as indicated on the right:

```
ABC E, 'ALFA', S'1          SET B1, 'ALFA'
                            GOMI E
                            RETURN
```


Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

```
L  :  PROC                          L  :  PROC
      ABC, E, 'B', S'T                      SET B'T, 'B'
      END                                   GOMI E
                                            RETURN
                                            END

      QX SZ'3, 4, X                         CLEAR I4
                                            ABC X, '+', S'3
```

and then by

```
                                            CLEAR I4
                                            SET B3, '+'
                                            GOMI X
                                            RETURN
```

7.4. Block structure and the scope of declarations

7.4.1. Procedures: external and internal

EOL uses the concepts of external procedures and of nested block structure of PL/1 [4]. The main purpose of these concepts is to allow separate compilation of external procedures, and, within an external procedure, to delimit the scope of a declaration of an identifier to only those parts of a program where this identifier is used. This allows the same identifier (in the case of EOL: a label or a formal index) to denote different quantities in different parts of a program.

An EOL procedure is defined by an occurrence of the symbol PROC or MAIN and the corresponding occurrence of the symbol END (see 7.2.). The <procedure body> which occurs between the symbols PROC or MAIN and END that define a procedure P is said to be the content of P, and any part of this procedure body is said to be contained in P.

A procedure which is not contained in any other procedure (i.e., an "outermost" one) is an external procedure, and its procedure name as well as all its entry points are external labels.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Any other procedure is an internal procedure.

That part of the content of a procedure P that is not contained in any other procedure contained in P is said to be internal to P, and is called the interior of P.

7.4.2. Declaration of labels

A label is declared by occurring at the left of an EOL statement, i.e. an instruction, declaration or procedure. Occurrence on the right of a statement represents use of the label, not a declaration.

The name(s) of an external procedure and all its entry points (i.e. all the labels of entry declarations (see 6.3.) that occur in the interior of an external procedure are external labels. All other labels are internal.

7.4.3. Scope of a declaration of a label

The scope of a declaration of an external label is the content of the external procedure whose name or entry point this label is, excluding the contents of all procedures contained in (or equal to) this external procedure in whose interior another declaration of the same label occurs.

The scope of any declaration other than an entry declaration of an internal label is defined as the content of that procedure P to which the declaration is internal, excluding the contents of all procedures contained in P in whose interior another declaration of the same label occurs. The scope of an entry declaration of a label L is the same as if X had been declared to be a name of the procedure in whose interior this entry declaration occurs.

Anywhere within the scope of a declaration D of a label L, use of L refers to the same label, namely the one declared by D. If D is an external declaration (see 6.4.), then use of L refers to the external procedure whose name is L or which has an entry point L.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

A procedure name or entry point L of an external procedure can be referred to from another external procedure provided the place from which reference is made lies within the scope of an <external declaration> of the label L.

7.4.4. Scope of an index declaration

The scope of an index declaration (see 6.5.) is the content of that procedure P to which this declaration is internal, with the exclusion of the contents of all procedures contained in P in whose interior another declaration of the same formal index occurs.

7.4.5. Use of labels and formal indexes

Every use of a label or formal index must lie within the scope of exactly one declaration of that label or formal index, respectively. Otherwise, that label or formal index will be either undefined or multiply defined. There is no interaction between labels and formal indexes, i.e. it is possible that the declarations of a label X and of a formal index X have overlapping scopes.

For compilation of an external procedure the above requirement on the use of labels is sufficient. For successful execution of a program, on the other hand, it is required in addition that if a label L occurs in an external declaration, then there exists in the program an external procedure whose name is L or which has an entry point L.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

7.4.6. Example

Arrows indicate the declaration referred to by each use of a label.

```
X  :  PROC
      EXTERN  M
A  :  GOMI    M
      CALL    X
      GOTO    A
      END

M  :  MAIN
        :
X  :  PROC
        :
      END
Y  :  PROC
      EXTERN  X
      CALL    X
        :
      END
Z  :  PROC
      CALL    X
        :
      END
      END
```

7.5. Flow of control in a program

Execution of a program begins with the first instruction that is internal to the procedure declared to be MAIN (see 7.2.). With the exeception of jump instructions (see 5.5.), which specify their successor explicitly, control always passes to the next instruction that is internal to the same procedure as the present instruction is. This means that all declarations and all procedures contained in the procedure in whose interior control is at present

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

are skipped in the process of execution, and that the only way of entering or leaving a procedure is by means of a jump instruction. If the last instruction is the interior of a procedure is not a jump, or is a conditional jump whose condition is not satisfied, then the successor of this instruction is not defined.

In the example below the label of each instruction coincides with the order in which the program is executed.

```
      MAIN
X  :  PROC
3  :  MOVE A1, Z2, *1
4  :  RETURN
      END

T  :  TABLE 'A', 'B', 'C'
1  :  READ I7, B1, B
2  :  CALL X
5  :  WRITE Z2, Q6
N  :  TABLE 1, 2, 3
6  :  STOP
      END
```


Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

BIBLIOGRAPHY

[1] Newell, A. et al., "Information Processing Language-V Manual," second edition, Prentice Hall 1964.

[2] "COMIT Programmers Reference Manual," MIT Press 1962.

[3] Naur, P. et al., "Revised Report on the Algorithmic Language ALGOL 60," Comm. ACM, Vol. 6, No. 1, pp. 1-17; January 1963.

[4] "PL/I: Language Specifications," IBM Systems Reference Library, Form C28-6571-2; January 1966.

[5] Lukaszewicz, L., "EOL-A Symbol Manipulation Language," The Computer Journal, Vol. 10, No. 1, pp. 53-59; May 1967.

[6] Lukaszewicz, L., "Outline of EOL Language," File No. 733, Department of Computer Science, University of Illinois; June 1967.

[7] Lukaszewicz, L., Nievergelt, J., "EOL Programming Examples—A Primer," Report No. 242, Department of Computer Science, University of Illinois; September 1967.

[8] "The EOL Compiler," File No. ___, Department of Computer Science, University of Illinois; to appear.

[9] "The EOL Interpreter," File No. ___, Department of Computer Science, University of Illinois; to appear.

[10] Lukaszewicz, L., "Language X1,", File No. 734, Department of Computer Science, University of Illinois; June 1967.

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by INTERNET ARCHIVE

Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

APPENDIX I. SUMMARY OF EOL SYNTAX

1. Programs, Procedures, Macro-definitions

```
<program>::=[<macro-definition>]..[<external procedure>]..

<macro-definition>::={<label>:}..MACRO ␣ <formal parameter list> λ
                        <procedure body> END [ ␣ <comment string>] λ

<external procedure>::=<procedure>

<procedure>::={<label>:}..{PROC|MAIN} [ ␣ <comment string>] λ
                <procedure body> END [ ␣ <comment string>] λ

<procedure body>::=[<statement> λ]..
```

2. Statements

```
<statement>::={[<label>:]..<instruction>|
               <declaration>|
               <procedure>|
               <comment>}
```

3. Instructions

```
<instruction>::={<stack instruction>|
                 <input instruction>|
                 <output instruction>|
                 <file instruction>|
                 <jump instruction>|
                 <table instruction>|
                 <macro instruction>|
                 <code>}
```


Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

4. Declarations

```
<declaration>::={<switch>|<table>|
                 <external declaration>|
                 <entry declaration>|
                 <index declaration>}
```

5. Comments

```
<comment>::=C[OMMENT] ␣ <comment string>

<comment string>::=<an arbitrary sequence of characters starting
                    with a character different from : and where
                    the character λ does not occur>
```

6. Stack instructions

```
MOVE ␣ {A|B|Z}<n>, {A|B|Y|Z}<m>[,{<number of steps>|
                                  <constituent test>}]

EXCHANGE ␣ {B<n>|IAS}, {B<m>|IAS}

SET ␣ {B|Z}<n>, {<integer>|<literal}

CLEAR ␣ A<n>[,{<number of steps>|<constituent test>}]

TEST ␣ {A|B|Z}<n>[,<constituent test>]

{ADD|SUB|MULT|DIV} ␣ {A|B|Z}<n>,[{B|Z}<n>],{<integer>|{B|Z}<k>}

WORD ␣ {A|B|Z}<n>[,{B|Z}<m>]

NUMBER ␣ {A|B|Z}<n>[,{B|Z}<m>]

SPLIT ␣ {A|B|Z}<n>[,{A|B|Y|Z}<m>|
                ,[{A|B|Y|Z}<m>],{<number of steps>|<word test>}]

COMPRESS ␣ {A|B}<n>[,{B|Z}<m>|
                    ,[{B|Z}<m>],{<number of steps>|<word test>}]

COUNT ␣ {A|B}<n>[,{B|Z}<m>|,[{B|Z}<m>],<constituent test>]
```


Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9y993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

7. Input instructions

READ ␣ I<n>,{B|Z}<m>[,{<number of steps>|<word test>}]

CLEAR ␣ I<n>[,{<number of steps>|<word test>}]

TEST ␣ I<n>[,<word test>]

8. Output instructions

WRITE ␣ {A|B|Z}<n>, Q<m>[,{<number of steps>|<word test>}]

SET ␣ Q<n>, <literal>

BLANK ␣ Q<n>, <number of steps>

LINE ␣ Q<n>, <number of steps>

PAGE ␣ Q<n>, <number of steps>

9. File instructions

PUT ␣ {A|B|Z}<n>,{C|D}<m>[,{<number of steps>|<constituent test>}]

GET ␣ {C|D}<n>,{A|B|Y|Z}<m>[,{<number of steps>|<constituent test>}]

SAVE ␣ P<n>,{B|Z}<m>

RESTORE ␣ [{A|B|Z}<n>,] P<m>

TEST ␣ P<n>[,<record test>]

SHIFT ␣ P<n>[,{<number of steps>|<record test>}]

SHIFTC ␣ P<n>,{B|Z}<m>[,{<number of steps>|<record test>}]

SET ␣ {C|D}<n>,{<integer>|<literal>|S*}

CLEAR ␣ C<n>[,{<number of steps>|<record test>}]

COPY ␣ {C|D}<n>,{C|D}<m>[,{<number of steps>|<record test>}]

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

10. Jump instructions

{GOTO|CALL|GOMI|GOPL|CAMI|CAPL} ␣ <label>

{GONA|GOIN|CANA|CAIN} ␣ {A|B|Z}<n>, <label>

RETURN

STOP

11. Table instructions

SEARCH ␣ {A|B|Z}<n>, <label>

SEARCHC ␣ {A|B|Z}<n>,[{B|Z}<m>], <label>

EXTRACT ␣ {A|B|Z}<n>,[{B|Z}<m>], <label>

12. Macro instructions

<macro instruction>::=<label> ␣ <actual parameter list>

13. Code

<code>::=CODE ␣ [<n>,[<m>,]]<label>

14. Switches

{<label>:}..{NAME|INDEX} ␣ <label>[,<label>]..

15. Tables

{<label>:}..TABLE ␣ {<literal> [,<literal>]..|
<integer> [,<integer>]..}

16. External declarations

{<label>:}..EXTERN [␣ <comment string>]

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

17. Entry declaration

{<label>:}..ENTRY [␣ <comment string>]

18. Index declaration

{<formal index>:}..EQU ␣ <actual index>

19. Labels

<label>::=<a non-empty sequence of characters different from
{ ␣ |,|:|'}>

20. Literals

<literal>::={'<any string of characters with the restriction
that quotation marks can occur only in adjacent
pairs. Such a pair inside the literal denotes
one quotation mark>' |{L|G}*}

21. Indexes

<actual index>::={0|1|2| ...|31}

<formal index>::=<letter>[<letter>|<digit>]..

22. Abbreviation for indexes

{<n>|<m>|<k>}::=<index>

23. Special symbols

<special symbol>::={L|G|S|R}*

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

24. Tests

<record test>::={<constitutent test>|{S|R}*}

<constituent test>::={<word test>|<number test>|<address test>

<word test>::={<class>|[PR|PE|FL|FE>/]{{B|Z|T}<k>|<literal>}}

<number test>::=[{LT|LE|GT|GE}/]{{B|Z|T}<k>|<integer>}

<address test>::={B|Z|T}<k>

<class>::=[L] [D] [B] [R]

25. Number of steps

<number of steps>::=*{<integer>|{B|Z}<k>}

26. Integers

<integer>::=[+|-]{digit}..

27. Parameter lists

<formal parameter list>::=[<formal parameter>[,<formal parameter>]..]

<actual parameter list>::=[<actual parameter>[,<actual parameter>]..]

28. Formal parameters

<formal parameter>::={[<macro name continuation>']<formal index>|
<formal label>|<formal literal>}

<macro name continuation>::=<letter> [<letter>|<digit>]..

<formal label>::=<label>

<formal literal>::=<literal>

29. Actual parameters

<actual parameter>::={[<macro name continuation>']{<actual index>|
<formal index>}|<label>|<literal>}

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

APPENDIX II. INDEX OF TERMS

A 4. 2. 3. 2. 1
A 4. 2. 3. 2. 2
ACTUAL INDEX 4. 2. 3. 3
ACTUAL PARAMETER 5. 7
ACTUAL PARAMETER LIST 5. 7
ADD 5. 1. 6
ADDING 4. 2. 3. 2. 2
ADDRESS 1. 2. 2
ADDRESS 3. 3. 3
ADDRESS TEST 4. 2. 4. 2
ADDRESS TEST 4. 2. 4. 2. 5
ARGUMENT 4. 2
ARITHMETIC INSTRUCTION 5. 1. 6
B 4. 2. 3. 2. 1
B 4. 2. 3. 2. 2
B 4. 2. 3. 2. 3
B 4. 2. 4. 2. 1
BASIC SYMBOL 2
BLANK 5. 3. 3
BLOCK STRUCTURE 7. 4
C 4. 2. 3. 2. 1
C 4. 2. 3. 2. 2
C 7. 2. 1
CAIN 5. 5. 3
CALL 5. 5. 1
CAMI 5. 5. 2
CANA 5. 5. 3
CAPL 5. 5. 2
CHARACTER 2. 1
CHARACTER STRING 3. 1
CLASS 4. 2. 4. 2. 1
CLASS LETTER 4. 2. 4. 2. 1
CLEAR FILE 5. 4. 6
CLEAR INPUT 5. 2. 2
CLEAR STACK 5. 1. 4
CODE 5. 8
COMMENT 7. 2. 1
COMMENT STRING 7. 2. 1
COMPRESS 5. 1.10
CONDITIONAL JUMP 5. 5. 2
CONSTITUENT 3. 3
CONSTITUENT COMPARED 4. 2. 4. 2. 4
CONSTITUENT TEST 4. 2. 4. 2
CONTAINED 7. 4. 1
CONTENT 7. 4. 1
CONVERT TO NUMBER 5. 1. 8
CONVERT TO WORD 5. 1. 7
COPY 5. 4.10

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

COUNT 5. 1.11
D 4. 2. 3. 2. 1
D 4. 2. 3. 2. 2
D 4. 2. 4. 2. 1
DECLARATION 6
DECLARATION OF FORMAL INDEXES 6. 5
DECLARATION OF LABELS 7. 4. 2
DELIMITER 2. 1. 3
DIGIT 2. 1. 2
DIV 5. 1. 6
ENTRY 6. 3
ENTRY DECLARATION 6. 3
EQU 6. 5
EXCHANGE 5. 1. 2
EXTERN 6. 4
EXTERNAL DECLARATION 6. 4
EXTERNAL LABEL 7. 4. 1
EXTERNAL PROCEDURE 7. 1
EXTERNAL PROCEDURE 7. 4. 1
EXTRACT 5. 6. 3
FE 4. 2. 4. 2. 2
FILE 1. 2. 5
FILE 3. 4
FILE INSTRUCTION 5. 4
FILE VALUE 3. 4
FL 4. 2. 4. 2. 2
FLOW OF CONTROL 7. 5
FORMAL INDEX 4. 2. 3. 3
FORMAL INDEX 7. 3
FORMAL LABEL 7. 3
FORMAL LITERAL 7. 3
FORMAL PARAMETER 7. 3
FORMAL PARAMETER LIST 7. 3
G* 4. 2. 1
GE 4. 2. 4. 2. 3
GET 5. 4. 2
GOIN 5. 5. 3
GOMI 5. 5. 2
GONA 5. 5. 3
GOPL 5. 5. 2
GOTO 5. 5. 1
GT 4. 2. 4. 2. 3
H LOGICAL REGISTER 1. 2. 4
I 4. 2. 3. 2. 1
IAS 1. 2. 3
INDEX 4. 2. 3. 3
INDEX 6. 1. 2
INDEX DECLARATION 6. 5
INDEX SWITCH 6. 1. 2
INDEXED MODE SYMBOL 4. 2. 3
INITIAL VALUES 3. 6
INPUT 1. 2. 1
INPUT 3. 1
INPUT INSTRUCTION 5. 2
INPUT VALUE 3. 1
INSTRUCTION 5
INSTRUCTION ADDRESS 3. 3. 3. 1

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

INSTRUCTION ADDRESS STACK 1. 2. 3
INSTRUCTION ADDRESS STACK 3. 3. 5
INTEGER 3. 3. 2
INTEGER 4. 2. 2
INTERIOR 7. 4. 1
INTERMEDIATE LIST 4. 2. 3. 1
INTERNAL 7. 4. 1
INTERNAL PROCEDURE 7. 4. 1
JUMP INSTRUCTION 5. 5
K 4. 2. 3. 3
L 4. 2. 4. 2. 1
L* 4. 2. 1
LABEL 4. 2. 5
LE 4. 2. 4. 2. 3
LETTER 2. 1. 1
LINE 5. 3. 3
LITERAL 4. 2. 1
LOGICAL REGISTER H 1. 2. 4
LOGICAL VALUE 3. 5
LT 4. 2. 4. 2. 3
M 4. 2. 3. 3
MACRO DEFINITION 7. 3
MACRO INSTRUCTION 5. 7
MACRO NAME 5. 7
MACRO NAME 7. 3
MACRO NAME CONTINUATION 7. 3
MAIN 7. 2
MARKER 2. 2
MODE OF ADDING 4. 2. 3. 2. 2
MODE OF READING 4. 2. 3. 2. 1
MODE OF TESTING 4. 2. 3. 2. 3
MODE SYMBOL 4. 2. 3. 2
MOVE 5. 1. 1
MULT 5. 1. 6
N 4. 2. 3. 3
NAME 6. 1. 1
NAME SWITCH 6. 1. 1
NORMAL RECORD 1. 2. 5
NUMBER 1. 2. 2
NUMBER 3. 3. 2
NUMBER OF STEPS 4. 2. 4. 1
NUMBER TABLE 6. 2. 2
NUMBER TEST 4. 2. 4. 2
NUMBER TEST OPERATOR 4. 2. 4. 2. 3
OPERATOR 4. 1
OUTPUT 1. 2. 1
OUTPUT 3. 2
OUTPUT FORMAT INSTRUCTION 5. 3. 3
OUTPUT INSTRUCTION 5. 3
OUTPUT VALUE 3. 2
PAGE 5. 3. 3
PE 4. 2. 4. 2. 2
POINTER 1. 2. 5
POINTER POSITION 3. 3. 3. 2
PR 4. 2. 4. 2. 2
PRECEDENCE OF CHARACTERS 2. 1. 4
PRECEDENCE OF WORDS 3. 3. 1. 1

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

PROC 7. 2
PROCEDURE 7. 2
PROCEDURE BODY 7. 2
PROCEDURE NAME 7. 2
PROCEDURES 7. 4. 1
PROGRAM 7. 1
PUT 5. 4. 1
Q 4. 2. 3. 2. 2
R 4. 2. 4. 2. 1
R* 4. 2. 4. 3
READ 5. 2. 1
READING 4. 2. 3. 2. 1
READING A FILE 4. 3. 2
READING A STACK 4. 3. 1
READING AN INPUT 4. 3. 1
RECORD 1. 2. 5
RECORD 3. 4
RECORD TEST 4. 2. 4. 3
REFERENCE LANGUAGE 1. 5
REPRESENTATION OF REFERENCE LANGUAGE 1. 5
RESTORE 5. 4. 4
RETURN 5. 5. 4
S* 4. 2. 4. 3
SAVE 5. 4. 3
SCOPE OF A DECLARATION OF A LABEL 7. 4. 3
SCOPE OF AN INDEX DECLARATION 7. 4. 4
SCOPE OF DECLARATIONS 7. 4
SEARCH 5. 6. 1
SEARCH AND COUNT 5. 6. 2
SET INTO FILE 5. 4. 5
SET INTO STACK 5. 1. 3
SET OUTPUT 5. 3. 2
SHIFT 5. 4. 8
SHIFT AND COUNT 5. 4. 9
SIMPLE JUMP 5. 5. 1
SPECIAL RECORD 1. 2. 5
SPLIT 5. 1. 9
STACK 1. 2. 2
STACK 3. 3
STACK INSTRUCTION 5. 1
STACK VALUE 3. 3
STATEMENT 7. 2
STOP 5. 5. 5
SUB 5. 1. 6
SWITCH 6. 1
SWITCH JUMP 5. 5. 3
SYNTAX FORMAL DESCRIPTION 1. 4
T 4. 2. 3. 2. 3
TABLE 6. 2
TABLE INSTRUCTION 5. 6
TERMINATION OF READING A LIST 4. 3
TEST 4. 2. 4
TEST FILE 5. 4. 7
TEST INPUT 5. 2. 3
TEST STACK 5. 1. 5
TESTING 4. 2. 3. 2. 3
USE OF FORMAL INDEXES 7. 4. 5

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

USE OF LABELS 7. 4. 5
VARIABLE 3
WORD 1. 2. 2
WORD 3. 3. 1
WORD TABLE 6. 2. 1
WORD TEST 4. 2. 4. 2
WORD TEST OPERATOR 4. 2. 4. 2. 2
WRITE 5. 3. 1
Y 4. 2. 3. 2. 2
Z 4. 2. 3. 2. 1
Z 4. 2. 3. 2. 2
Z 4. 2. 3. 2. 3

SYMBOLS

BRACES 1. 4
BRACKETS 1. 4
CONSTITUENT MARK FOR ADDRESS 1. 2. 2
CONSTITUENT MARK FOR NUMBER 1. 2. 2
CONSTITUENT MARK FOR WORD 1. 2. 2
END OF TEXT MARKS 1. 6
GAMMA 2. 1. 3
LAMBDA 1. 5
LAMBDA 2. 1. 3
POINTER 1. 2. 5
RECORD MARK 1. 2. 5
SIGMA 1. 2. 5
SIGMA 3. 4
SLASH 1. 4

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0
Digitized by INTERNET ARCHIVE
Original from UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

AUG 16 1968

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

JUN 20 1969

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN

UNIVERSITY OF ILLINOIS-URBANA
510.84 IL6R no. C002 no.237-242(1967
Parallelism exposure and exploitation in
3 0112 088398331

Generated on 2023-08-01 17:45 GMT / https://hdl.handle.net/2027/uiuo.ark:/13960/t7xk9v993
Creative Commons Attribution-NonCommercial-ShareAlike / http://www.hathitrust.org/access_use#cc-by-nc-sa-3.0

Digitized by
INTERNET ARCHIVE

Original from
UNIVERSITY OF ILLINOIS AT
URBANA-CHAMPAIGN