AN OPERATING SYSTEM FOR THE LINC COMPUTER
Richard Kelly Moore

Technical Report No. IRL-1038
November 1, 1965

Prepared under:
National Aeronautics and Space Administration
Grant NsG 81-60
National Institute of Health
Grant FR 00151-01
Air Force Office of Scientific Research
Contract AF 49 (638)-1599
Grant 886-65

Principal Investigator: J. Lederberg
Program Director: E. Levinthal

Instrumentation Research Laboratory, Department of Genetics
Stanford University School of Medicine
Palo Alto, California
FOREWORD

The Instrumentation Research Laboratory was assigned a LINC computer August 9, 1963 as a participant in the LINC Evaluation Program under the sponsorship of the NIH, NASA and U.S. Air Force. We found LINC to be an extremely valuable adjunct to laboratory instrumentation. However, our environment may differ from that of most LINC users in the unpredictable variety of applications and we found a large proportion of our effort going into the programming of the computer and into linking subroutines with one another. This justified the effort described herein to provide an operating system with some higher level language capability. Part of this operating system is "BLINK" an algorithmic compiler which is run, not on LINC but on an IBM-7090. However, other aspects of the system are already self-contained in the LINC and may therefore be helpful to LINC users even without ready access to the larger computer. BLINK may also be a vehicle for writing other augmentations of the LINC operating system, just as Mr. Moore has used it to generate the facilities described here. Alternatively, some of our colleagues may now be stimulated to write a LINC-code-generator (analogous to BLINK) in a more generally available language like FORTRAN, which would widen the range of background machines to support programming efforts for running on a LINC embedded, as it was designed to be, in a laboratory environment.

This system can be made even more effective by direct wire communication between the LINC and the central computer. Pending the installation of comprehensive time-sharing, the job priority sequence of the Stanford IBM-7090 cannot be interrupted. However, facilities exist for access to a disc file shared by the 7090 and a PDP-1 computer. As time sharing is available on the latter, remote wire communication is possible between the LINC and a common
fast-access memory. A system is now being debugged for the exchange of BLINK texts from the LINC, stored on disc, compiled in proper course by the 7090, the assembly then waiting on disc to be called by LINC for loading. This may be regarded as a rather clumsy prototype of a system in which a time-shared central computer has, as one of its most important roles, the programming of peripheral computers which must remain in uninterrupted contact with their assigned tasks.

We have found this separation of programming from execution functions to be a powerful approach that deserves general consideration in evaluating the role of smaller computers. It perhaps does have its main use in the more complicated system and utility programs. Simpler routines and patches can still be inserted at lower levels, closer to the machine itself.

J. Lederberg
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page No.</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. Introduction to LOSS System</td>
<td></td>
</tr>
<tr>
<td>A. Foreward</td>
<td>11</td>
</tr>
<tr>
<td>B. Linc Hardware Description</td>
<td>1</td>
</tr>
<tr>
<td>C. Linc Operating System</td>
<td>3</td>
</tr>
<tr>
<td>II. Description of Basic Programs</td>
<td></td>
</tr>
<tr>
<td>A. Naming of Files</td>
<td>10</td>
</tr>
<tr>
<td>B. Text Editing</td>
<td>12</td>
</tr>
<tr>
<td>C. File Listing</td>
<td>16</td>
</tr>
<tr>
<td>D. Automatic Input-Output</td>
<td>18</td>
</tr>
<tr>
<td>E. Linc-Tape Display</td>
<td>19</td>
</tr>
<tr>
<td>F. IBM-Tape Display</td>
<td>22</td>
</tr>
<tr>
<td>G. Loading of Blink Compilations</td>
<td>25</td>
</tr>
<tr>
<td>H. Symbolic Assembler</td>
<td>26</td>
</tr>
<tr>
<td>III. Use of In-Core LOSS Features</td>
<td></td>
</tr>
<tr>
<td>A. Overlay Stack</td>
<td>39</td>
</tr>
<tr>
<td>B. Buffer</td>
<td>40</td>
</tr>
<tr>
<td>C. Available Routines</td>
<td>41</td>
</tr>
<tr>
<td>D. System Compatibility</td>
<td>42</td>
</tr>
<tr>
<td>E. Automatic Buffer Dump</td>
<td>43</td>
</tr>
<tr>
<td>IV. Blink</td>
<td></td>
</tr>
<tr>
<td>A. Language Description</td>
<td>44</td>
</tr>
<tr>
<td>B. Input-Output Techniques</td>
<td>51</td>
</tr>
<tr>
<td>Appendix A: Detailed Description of LOSS System</td>
<td></td>
</tr>
<tr>
<td>1. Listing of Quarter 0</td>
<td>53</td>
</tr>
<tr>
<td>2. SAVE-RESTORE Stack</td>
<td>54</td>
</tr>
<tr>
<td>3. PARAMS: Transfer of Parameters to Procedures</td>
<td>54</td>
</tr>
</tbody>
</table>
4. REPEAT: Iterative Loops
5. PUT and GET: Use of Buffer
6. Buffer: Details of List Structure
7. OVERLAY: Program Loading
8. Files: Index and Control Words
9. Compilation of Blink Programs

Appendix B: Teletype Listing of System Features
1. Program Stack Contents
2. Assembly Listings of Basic Programs
3. File Indices
4. Text Listings (Symbolic Programs)

Appendix C: Blink Compilations
1. Assembler, Pass 1
2. Assembler, Pass 2-4
3. LOSS Monitor
   - Relocatable Octal Code
   - Symbol Table
4. Question-Answer
5. Define-Erase
6. Blink Loader
The Linc (Laboratory Instrument Computer)

The Linc is a general purpose, solid state, digital computer designed for high speed data handling in the laboratory. The Linc is a single address, fixed 12-bit word length, binary computer using 1's complement arithmetic and 2's complement arithmetic to facilitate multi-precision arithmetic. A typewriter-like keyboard and a program-controlled oscilloscope provide basic input-output facility. In addition, a six-bit relay register, 16 lines of analog-digital conversion, 2 digital-analog output channels, and 16 pulse lines are available to control or monitor external devices. Two tape units provide an addressable data storage area of up to 3,072,000 bits (256,000 12-bit words). With a basic cycle time of 8 micro-seconds, the Linc can perform 41,000 indexed additions per second. The 2048-word random access magnetic core memory is divided into 8 "quarters" of 256 words each. The lower 4 quarters can contain either executable instructions or data, whereas instructions cannot be fetched directly from the upper four memory quarters.

Stanford's Instrumentation-Research-Laboratory Linc Configuration

In addition to the full 2048-word memory, the Linc system at Stanford includes a teletypewriter (Model 33), a Datamec tape unit (Model D2020), a Calcomp digital plotter, and a special facility for executing instructions in the upper half of memory. The full-size memory and the teletype are minimal requirements for implementation of the Loss operating system. The Datamec tape unit and Calcomp plotter (together with a graph-reading photoelectric cell) greatly enhance the input-output flexibility of the system. The ability to execute instructions in upper core is not used in any of the Loss system programs, and its desirability has not been found to be of high priority.
The software systems in use include both Loss, a basic operating system for the Linc, and Blink, a compiler which runs on Stanford's IBM 7090 and which translates Algol-like programs into relocatable Linc code. Programs created with the Blink compiler are loaded onto Linc tape from the Datamec tape unit.

The system herein described was begun by the author in the Spring of 1964 under the direction of Dr. Joshua Lederberg, head of the Stanford Genetics Department. Final debugging and documentation were completed during the summer of 1965. The author wishes to express his sincere appreciation and gratitude to the following persons for their contributions to the completion of this project: Dr. Lederberg, for his patient support of a system long in reaching a state of utility; Lawrence Tesler, who contributed many of the basic ideas used; the Subalgol Compiler Group, especially Larry Breed, for their help in changing the Subalgol compiler into a Blink compiler.

This documentation does not include an introduction to the basic Linc instruction set or to the common techniques of Linc coding. The reader is referred to the excellent treatment by Mary Allen Wilkes and Wesley A. Clark (LINC-Vol. 16, Programming and Use-1, Washington University, St. Louis, Missouri, April-June, 1965).
The Linc Operating System (Loss)

Loss is a complete operating system for the Linc computer. Under Loss, Linc programs can be created, updated, and used. The basic components of this system are the monitor, files, the program stack, and various utility programs. The monitor is a supervisory program which governs the execution of all other programs; files are data storage areas which are identified by a name and a book number; the program stack is the library of available programs; and the utility programs are programs so frequently used that they are present on every program stack tape.

The Monitor

The monitor is loaded by the following procedure:

(1) Mount a Loss program stack tape onto unit 0.
(2) Read block 150 of the program stack tape into memory quarter 0 by setting the left and right switches to 0700 0150 and raising the DO-TOG lever.
(3) Begin execution at cell 0 by putting 0000 into the right switches and pressing the START RS button.

All further control of the monitor is through the on-line typewriter. At each step, the monitor informs the user of what his next action should be. The first message typed by the monitor is PROG, which requests that the user type in the name of a program to be executed. If the user types a name which is not the name of an available program, the monitor will type NO SUCH PROG, following which the user can try again.
If the user wishes to know which programs are available he can type the character ? rather than a program name, and the monitor will type a list of all programs presently on the program stack.

When the user has finally typed the name of an available program, the monitor types a message which describes the parameters required by the program.

The following kinds of parameters will be accepted:

1. Octal numbers (possibly preceded by a minus sign)
2. Alphabetic strings
   A string is entered by typing the character %; the first printing character thereafter becomes the delimiter of the string. All characters appearing after the first occurrence of the delimiter, but before its second occurrence, are included in the string. E.g., "ABC" contains just A, B and C.
3. Files
   A file is entered by typing the character /, followed by the number of the book containing the desired file, followed by a carriage return and the name of the file itself. If a ? is typed in place of a file name, the contents of the index of the referenced book will be typed out.

The typing of a ; signals the monitor that all necessary parameters have been supplied, and causes the requested program to be loaded. After typing the message RUN, the monitor reads the appropriate program stack location into core and execution begins. When the called program has completed its execution, control is automatically returned to the monitor.
Files

Under Loss, tape unit 1 of the Linc is a structured data storage area. This tape is divided into 10* books, each book consisting of 100 tape blocks. Book 0 contains blocks 0 through 77, book 1 contains blocks 100 through 177, etc. Each book is divided into files, each file being a group of consecutive tape blocks. The 0th block of each book is reserved as an index; the index contains a name, a length, and an initial block number corresponding to each file. Such a structured file system allows the user to refer to all data files by name, and affords a measure of over-write protection to stored data.

Some examples of the kinds of data which might conveniently be stored in files are (1) voltage readings from external devices, (2) results of arithmetic calculations, or (3) alphabetic information entered from the typewriter.

The Program Stack

Tape unit 0 is reserved by Loss for absolute, ready-to-load, Linc programs. This tape, as a whole, is called the program stack, and each group of 10* blocks is a program stack location. Program 1 occupies blocks 10 through 17, program 5 occupies blocks 50 through 57, etc. All programs which are available for use under Loss occupy some program stack position. Each of these programs is identified not only by its stack position, but also by a name which is stored in an index referred to by the monitor.

*All numbers in this document are in the octal system, with the exception of page headings and Blink examples.
An Example

In order to further describe and clarify the process of user-monitor communication, let us consider a call of the hypothetical program TYPE, a program which prints the contents of a variable number of files:

```
<PROG>>
*TYPER
<NO SUCH PROG
<PROG>>
*TYPE
<NO. OF FILES, FILES>>
*3
*PROGA
<BOOK UNDEF
*/1
*PROGA
*PROGB
*PROGC
<NO SUCH FILE
*/2
*PROGC
*
<RUN
```

As suggested by the descriptive message, the user first supplies the octal parameter 3 -- meaning that three files are to be printed -- and next enters the names of the three files of interest. Note that each item entered by the user must be followed by a carriage return, and that if consecutively referenced files are in the same book, it is not necessary to re-enter the book number.

The descriptive message associated with each program is meant only as a reminder of the required parameters; if the meaning of a message is unclear to the user, he should refer to a more detailed description of the particular program he is trying to use.

* In this document, an asterisk flags lines typed by the user, as opposed to lines typed by the computer.
Errors

Most errors which a user might commit are easily corrected under monitor control: errors which cause the computer to halt, or otherwise necessitate the reloading of the monitor, are almost non-existent. Some errors are detected by the monitor itself, as the preceding TYPE example shows. In such cases, the monitor types a message describing the error, and ignores the line on which the error occurred. If the user wishes to correct faulty input to the monitor, there are three levels of recovery available to him: (1) If the user detects an error on a line before he has typed a carriage return, he can erase the entire line by hitting the RUBOUT key. (2) If he detects an error before typing the terminating ;, the user can begin his program call completely anew by typing a period and a carriage return. (3) If all else fails, the user can halt the computer (by depressing the HALT lever) and reload the monitor through the switches.
<table>
<thead>
<tr>
<th>Stack Location Number</th>
<th>Program Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-7</td>
<td>(unused)</td>
</tr>
<tr>
<td>10</td>
<td>TYPE</td>
</tr>
<tr>
<td>11</td>
<td>BLINK</td>
</tr>
<tr>
<td>12</td>
<td>DISTAPE</td>
</tr>
<tr>
<td>13</td>
<td>DISPLAY</td>
</tr>
<tr>
<td>14</td>
<td>DEFINE</td>
</tr>
<tr>
<td>15</td>
<td>MONITOR</td>
</tr>
<tr>
<td>16</td>
<td>EDIT</td>
</tr>
<tr>
<td>17-22</td>
<td>(overlay stack)</td>
</tr>
<tr>
<td>23</td>
<td>QA</td>
</tr>
<tr>
<td>24</td>
<td>LASS</td>
</tr>
<tr>
<td>25</td>
<td>LASS (pass 2)</td>
</tr>
<tr>
<td>26-</td>
<td>(unused)</td>
</tr>
</tbody>
</table>
### LOSS Character Codes

<table>
<thead>
<tr>
<th>CHARACTER</th>
<th>CODE</th>
<th>CHARACTER</th>
<th>CODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>00</td>
<td>A</td>
<td>41</td>
</tr>
<tr>
<td>&quot;</td>
<td>01</td>
<td>B</td>
<td>42</td>
</tr>
<tr>
<td>#</td>
<td>02</td>
<td>C</td>
<td>43</td>
</tr>
<tr>
<td>$</td>
<td>03</td>
<td>D</td>
<td>44</td>
</tr>
<tr>
<td>%</td>
<td>04</td>
<td>E</td>
<td>45</td>
</tr>
<tr>
<td>&amp;</td>
<td>05</td>
<td>F</td>
<td>46</td>
</tr>
<tr>
<td>(</td>
<td>06</td>
<td>G</td>
<td>47</td>
</tr>
<tr>
<td>)</td>
<td>07</td>
<td>H</td>
<td>48</td>
</tr>
<tr>
<td>*</td>
<td>08</td>
<td>I</td>
<td>49</td>
</tr>
<tr>
<td>+</td>
<td>09</td>
<td>J</td>
<td>50</td>
</tr>
<tr>
<td>-</td>
<td>10</td>
<td>K</td>
<td>51</td>
</tr>
<tr>
<td>.</td>
<td>11</td>
<td>L</td>
<td>52</td>
</tr>
<tr>
<td>/</td>
<td>12</td>
<td>M</td>
<td>53</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
<td>N</td>
<td>54</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
<td>O</td>
<td>55</td>
</tr>
<tr>
<td>2</td>
<td>15</td>
<td>P</td>
<td>56</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
<td>Q</td>
<td>57</td>
</tr>
<tr>
<td>4</td>
<td>17</td>
<td>R</td>
<td>58</td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td>S</td>
<td>59</td>
</tr>
<tr>
<td>6</td>
<td>19</td>
<td>T</td>
<td>60</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>U</td>
<td>61</td>
</tr>
<tr>
<td>8</td>
<td>21</td>
<td>V</td>
<td>62</td>
</tr>
<tr>
<td>9</td>
<td>22</td>
<td>W</td>
<td>63</td>
</tr>
<tr>
<td>:</td>
<td>23</td>
<td>X</td>
<td>64</td>
</tr>
<tr>
<td>;</td>
<td>24</td>
<td>Y</td>
<td>65</td>
</tr>
<tr>
<td>&lt;</td>
<td>25</td>
<td>Z</td>
<td>66</td>
</tr>
<tr>
<td>=</td>
<td>26</td>
<td>(</td>
<td>67</td>
</tr>
<tr>
<td>&gt;</td>
<td>27</td>
<td>)</td>
<td>68</td>
</tr>
<tr>
<td>?</td>
<td>28</td>
<td>(end of string)</td>
<td>69</td>
</tr>
<tr>
<td>@</td>
<td>29</td>
<td>(end of text)</td>
<td>70</td>
</tr>
</tbody>
</table>

**Code derivation:**

LDA (teletype code)

GGM

ADA 1

0277

SCR 1
Program name: DEFINE
Parameters: 1 string

Description:
The define program is used to allocate storage space for files. As an example, the following program call creates a file named CHARLIE, of length 7 blocks, in book 6:

```
<PROG>>
* DEFINE
  1 STRING>
* %+ BOOK 6 DEFINE
* CHARLIE 7 +
* ;
<RUNO>>
```

(param request
(begin a string
(carr. ret. allowed

From now on, the reserved 7 blocks can be referenced by the symbols:

* /6
* CHARLIE

If, at some future time, the file CHARLIE is no longer needed, its tape blocks can be released by erasing its name from the index of book 6. The following call of the define program would effect this erasure:

```
<PROG>>
* DEFINE
  1 STRING>
* %$BOOK 6 ERASE CHARLIE$
* ;
<RUNO>>
```

In general, the following operations are accepted:

- **BOOK n** : what follows modifies Index of book n
- **DEFINE name length** : create a file, reserving tape space
- **ERASE name** : destroy a file, freeing tape space

10.
The DEFINE program scans its input string for occurrences of the preceding operations; as each operation is interpreted, an appropriate modification is made to the specified index. Any number of operations can be included in a single call of DEFINE, any number of books can be affected, and any number of spaces or carriage returns can separate items of the string.

An appropriate error message is typed if the book specification is absent, if an attempt is made to erase a non-existent file or to define a file with an already used name, or if storage capacity of a book is exceeded.
Program name: EDIT
Parameters: 1 file

Description:

The edit program (the editor) is used to create and to update alphabetic files (texts). The editor displays on the scope a portion of the text being edited, and simultaneously accepts instructions from the typewriter as to how the text should be modified. At any given time, a unique line of the text is defined to be the line of attention; this line is always the bottom line being displayed by the editor, and, for clarity, it is underlined on the scope.

The top line of the display is not actually a line of the text, but rather contains two descriptive numbers: the first (in parenthesis) is the number of blocks occupied by the text in its current form, the second is the line number of the line of attention.

There follows a list of the control characters (operations) which are recognized by the editor. The explanations strive to be complete, but the user need not memorize them before he can use the editor effectively: since the results of each operation are immediately displayed on the scope, the user can efficiently become familiar with the editor through trial and error, referring to the write-up only when necessary.
Abbreviations:

- n : an octal number
- Z : a carriage return
- μ : an arbitrary character (constant within a single operation)
- θ : an arbitrary string of characters
- s : a space

<table>
<thead>
<tr>
<th>Command</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>nA or nB</td>
<td>Line number n becomes the line of attention. Any value of n which exceeds the number of lines in the text will move attention to the end of the text: the end-of-text symbol (a solid square) will be displayed, alone, as the line of attention.</td>
</tr>
<tr>
<td>nA (Ahead)</td>
<td>The line of attention is advanced n lines.</td>
</tr>
<tr>
<td>nB (Back)</td>
<td>The line of attention is moved back n lines.</td>
</tr>
<tr>
<td>A(θ)</td>
<td>The display is moved ahead (back) using the same value of n last used with either an A or B command. If not otherwise specified n is one (1).</td>
</tr>
<tr>
<td>FθBu (Find)</td>
<td>(where θ does not contain μ) The text is searched (beginning at the line of attention) for an occurrence of θ. If an occurrence is found, the line of attention is advanced so that θ appears on the screen. If no occurrence is found, attention is set at the end of the text.</td>
</tr>
<tr>
<td>FθBu</td>
<td>The next occurrence of θ is sought. Thus the following sequence of commands would cause the fourth occurrence of the word JAZZ to be displayed (or else the end-of-text symbol):</td>
</tr>
<tr>
<td>nD (Delete)</td>
<td>n lines are deleted from the text, beginning with the line of attention. If n is absent, only the line of attention is deleted.</td>
</tr>
</tbody>
</table>
| I (Input) | This command allows the user to insert text preceding the current line of attention. The editor types the character <, and displays a blank line as the line of attention. As the user enters characters on the typewriter, they appear on the scope on the line of attention. The user can erase the last character
on the line he is filling by hitting RUBOUT. When the user types a carriage return, the just-filled line moves up one position on the scope, the line number increases by 1, and the editor prepares for the next line of input. A < is typed and a blank line again appears on the screen. When the user has finished typing the last line of input, with its terminating carriage return, he types an exclamation point, !, which returns the editor to control mode. At that time, the former line of attention will re-appear on the screen, preceded by the inserted lines. If the line of attention was the end-of-text symbol, then the added lines will be the last lines of the text. If the capacity of the text is ever exceeded by an input line, the editor will type the message *FULL and automatically return to control mode.

The maximum length text which can be processed by the editor is four (4) blocks.

**(C)** (Change)

The line of attention is replaced in the text by the next line entered from the typewriter.

**(Sub)** (Substitute) (where neither $\beta_1$ nor $\beta_2$ contain $\mu$)

The text, beginning at the line of attention, is searched for occurrences of $\beta_1$; every such occurrence is replaced in the text by $\beta_2$. The editor types out the number of substitutions made and sets attention at the end of the text. $S$ is a very powerful editing tool; here are some examples of its use:

1. **$S$JMP$S$JMP$**
   this operation counts the number of times the string JMP appears in the text.

2. **$S$O.LDAYQyJMP PARAMS.**
   Here, $S$ is used essentially as a macro generator for the creation of an assembly program. The single letter Q was typed everywhere in a text where the longer string would eventually be needed.

3. **$S$/RECEIVE/RECEIVE/**
   Here, a spelling error is corrected.
L (Line)  The current line number is typed out.
T (Type)  n lines are typed out, beginning with the
          line of attention.
R (Read)  The text being edited is re-read from tape.
          This operation can be used to recover from
          reckless editing.
W (Write) The text being edited is written on tape.
M (Monitor) The text is written on tape, and control is
            returned to the monitor.

Errors

Hitting RUBOUT while in control mode will abort any
unfinished command and will cause the editor to type two
question marks, ??, and a carriage return. The editor
will type the same sequence, ??γ, whenever it encounters
an illegal command.
Program name: TYPE

Parameters: MSG, Format no., initial block, no. of blocks

This program prints the requested tape blocks on the on-line typewriter according to the specified format. If the desired tape blocks constitute a file, then the file name, when given to the monitor as a parameter, will automatically place both the block number and length on the buffer.

The format is specified by a single octal parameter and can have the following values:

format = 1:

The specified blocks are to be interpreted as a file (necessarily on tape unit 1) which contains the conventional control word.* The number of blocks typed will be the minimum of the length supplied as a parameter and the length indicated in the control word. If the file is designated as alphabetic in the control word, the file will be typed as an alphabetic text; otherwise the contents of the file will be typed as octal numbers.

format = 2:

The specified blocks are located on unit 1 and are to be typed as octal numbers. The first word of the tape area is not to be interpreted as a control word.

format = 3:

The specified blocks are on unit 0 and are to be typed in the octal format.

*See page 60.
If sense switch 0 is on, output pauses so that the external device (typewriter or analog recorder) can be adjusted. Sense switch 1 causes premature termination and return to monitor.

The first parameter, MSG, is a string which is printed on each page of output as a label.

More than one set of parameters can be specified at a single call of TYPE. An example of the call follows.

```
<PROG>
* TYPE
* /0
* "$NAME1:9/6/65"
* 1
* NAME1
* "$NAME2:9/6/65"
* 2
* NAME2
* "$BLOCK 322 ETC."'
* 2
* 322
* 4
* "$DATA"
* 3
* 0
* 1
* ;
<RUN>
```

Normally, output from the TYPE program is sent directly to the on-line teletype. However, in order to save computer time, the data can be sent at high speed to an analog tape recorder. In this case, sense switch 2 is turned on (signalling the computer to supply output at 8 times the normal rate) and the tape recorder is set at high speed and connected to the teletype line. The recorder is later set to low speed and "played back" into an off-line teletype.
Program name: QA (Question and Answer)
parameters: any record.

QA prints the top record of the buffer and replaces it by a record entered on the typewriter.

The formats in which items are printed and entered is precisely the same as with the monitor itself: the user can enter file names, strings, or octal constants.

This program is intended to be used as a dynamic input routine by other Loss programs: A running program puts a question (probably in the form of a string) on the buffer and overlays QA. QA prints the question, accepts the answer (a list of parameters) from the user, places the record on the buffer, and returns to the original program. The parameters can then be extracted from the buffer in the same way that parameters are accepted from the monitor.
Program name: DISPLAY
Parameters: none

This program displays any desired block of Linc tape in any one of three (3) formats. The block number is selected with pots 0 and 2 (left pot is coarse adjustment, right pot is fine adjustment) and sense switch 0 causes the specified block to be read. The highest order digit of the block number selects the tape unit: blocks 0-777 are on unit 0, 1000-1777 are on unit 1. The first line of the display always includes both the number of the block currently in core, and the block number selected to be read next.

Pots 4 and 6 select the line which is to be displayed. This line number, whose meaning varies according to the selected format, is the last number displayed on the first line of the scope.

Alphabetic format (sense switch 2 on):

Each half-word of the data block is interpreted as an alphabetic character according to the Loss character code. Lines are terminated by the code 73g, and the data block as a whole is terminated by 74g.

Texts created by Edit can thus be examined by Display without the necessity of returning to the monitor each time a
different text is desired. Of course the user cannot reference files by name or alter files while using Display.

**Index format (sense switch 3 on):**

The data block is expected to be a book index (the block number should therefore be either 1000, 1100, 1200, ... , or 1700). Each file entry is displayed on a separate line, the first file of the display being determined by the selected line number.

**Octal format (sense switch 4 on):**

The contents of the data block is displayed as octal numbers. The number of digits to be included on a line is selected by pot 3. Below each pair of octal digits is displayed the corresponding alphabetic character.

Any number of half words can be ignored at the beginning of the data block. The number to be skipped is selected by pots 5 and 7; their value (if non-zero) appears at the top of the scope following the word SKIP. The display and line numbering behave as if the data block began just after the skipped half words.
The skip feature enhances the formatting flexibility of Display. One example of its use would be the examination of the first block of an alphabetic text. In order that the display not be confused by the control word at the beginning of the block, pots 5 and 7 should be set to the value two (2): SKIP 2 will appear at the top of the screen, and the control word (2 half-words) will not be represented on the scope. When the line number is then set to one (1), the first character to appear on the screen (apart from the heading) will be the actual first character of the text (i.e., the third half-word of the file).

Sense switch usage:

<table>
<thead>
<tr>
<th>sense switch</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read selected block</td>
</tr>
<tr>
<td>1</td>
<td>return to monitor</td>
</tr>
<tr>
<td>2</td>
<td>alphabetic format</td>
</tr>
<tr>
<td>3</td>
<td>index format</td>
</tr>
<tr>
<td>4</td>
<td>octal format</td>
</tr>
</tbody>
</table>

Potentiometer summary:

<table>
<thead>
<tr>
<th>pot</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>block number</td>
</tr>
<tr>
<td>3</td>
<td>octal word size</td>
</tr>
<tr>
<td>4-6</td>
<td>line number</td>
</tr>
<tr>
<td>5-7</td>
<td>skip parameter</td>
</tr>
</tbody>
</table>

21.
Program name:  DISTAPE
Parameters:  none

This program displays on the Linc oscilloscope the contents of an IBM-compatible magnetic tape. The tape reel of interest should be mounted on the Datamec tape unit, the density switch appropriately set, and the speed correctly chosen: high-density tapes (556 b.p.i.) should be read at low speed (4.5 in./sec.) -- low density tapes (200 b.p.i.) at high speed (45 in./sec.).

The information on the tape is expected to be in the form of physical records none of whose lengths exceeds 1536 characters. Only the first 1536 characters of longer records will be displayed.

By use of the potentiometers and sense switches, the user specifies to the Distape program (1) the density of the tape being examined, (2) which physical record should be read into core, (3) what portion of the specified record should actually appear on the scope, and (4) what format the data should be displayed in.

Initially, Distape reads the first record in the forward direction. Pots 0 and 2 are then used to select the next record to be read. (The left pot is the coarse adjustment, the right pot is the fine adjustment). As these pots are adjusted, their combined value (either a positive or negative octal number) is displayed at the top of the scope. If this record selection number is one (1), then when sense switch 0 is momentarily lifted, the next tape record (in a forward direction) will be
read into core. In general, if the record selection number is n, then
n-1 records will be skipped, and the n'th record from the current pos-
tion will be read into core when sense 0 is next raised. In order to
successively view sequential tape records, the user sets the record
selection number to one (1): each time sense 0 is turned on, the next
record will be read. The tape can be spaced backwards by specifying a
negative record selection number. Minus one (-1), e.g., denotes
the record just previous to the current record.

When the desired record has been read into core, its contents
can be displayed either as BCD (alphabetic) information, or else as
binary data. (sense switch 3 on selects binary.) In the BCD mode,
the tape is expected to contain standard IBM BCD records with the
special character $32_8$ denoting a carriage return. Lines, which
may be of variable length, are displayed beginning at the left hand
group. Adjacent blank characters are telescoped into a
single space so that more information can be placed on the screen.
Pots 4 and 6 are set by the user to the desired line number; this line
number is displayed on the scope next to the record selection number.
The second line of the screen begins the display of the actual selected
data.
In the binary mode, each 6-bit byte of the record is interpreted as two octal digits. Pot 3 selects the number of bytes (up to 8) to be included on a single line of the display. When the size of a line has been thus defined, pots 4 and 6 select the line to be displayed. Immediately below each pair of octal digits is displayed the corresponding BCD character.

Sense switch usage:

<table>
<thead>
<tr>
<th>sense switch</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>read selected record</td>
</tr>
<tr>
<td>1</td>
<td>return to monitor</td>
</tr>
<tr>
<td>2</td>
<td>density selection (on=hi)</td>
</tr>
<tr>
<td>3</td>
<td>format selection (on=binary)</td>
</tr>
</tbody>
</table>

Potentiometer summary:

<table>
<thead>
<tr>
<th>pot</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-2</td>
<td>record selection number</td>
</tr>
<tr>
<td>3</td>
<td>octal word size</td>
</tr>
<tr>
<td>4-6</td>
<td>line number</td>
</tr>
</tbody>
</table>
Program name: BLINK
parameters: 1 file

This program searches (in a forward direction) the tape mounted on
the Datamec tape unit for an occurrence of a Blink compilation. The
relocatable code of the first such compilation is loaded onto the indicated
file and prefixed by a control word specifying type B'. The file is then
ready to be assembled by Lass together with other Blink programs or
symbolic assembly programs.

The Datamec tape should initially be at the load point. BLINK is
then called once for each compilation present on the tape reel, all tape
motion being controlled by BLINK. The contents of the tape can be examined
using the DISTAPE program, but the reel should then be rewound before BLINK
is called.

BLINK prints an appropriate message telling whether an apparent tape
parity error was encountered or whether the file was of sufficient length.
If the tape runs away, i.e., if no Blink compilation is encountered, then
mechanical intervention by the user is required for termination.

Sense switch 2 selects tape density (on=high), and sense switch 1
returns to monitor.
Program name: LASS
Parameters: stack number, files

Description:

This program is the Loss assembler. The referenced files -- alphabetic texts (symbolic assembly programs) and Blink files (relocatably compiled BalgoI-like programs) -- are combined to form an absolute Linc program which is placed at the indicated position on the program stack.

Symbolic Assembly Programs

Texts created by the editor (the program EDIT) can be converted by Lass into absolute Linc programs. Each line of such a text is interpreted by Lass either to be a meta-command to Lass itself (a pseudo-op), or else to represent a single word of Linc code (a code line). The pseudo-ops recognized by Lass are GLOBAL, UPPER, DEFINE, ORG, DITTO, VARB, and CONT: any line beginning with one of these words will be interpreted as a pseudo-op; all other lines are code lines.

code lines and expressions

Any word (a sequence of contiguous letters and digits beginning with a letter) which is not one of the seven pseudo-ops is interpreted by Lass to be an identifier. Every identifier is assigned a unique numerical value. Certain identifiers, such as those corresponding to Linc instructions (JMP, STC, LDA, etc.), are inherently assigned values by Lass (6000, 4000, 1000, etc.). All other identifiers must be given values through the DEFINE pseudo-op (to be explained later) or else through being used as the symbolic address
of a code-line: if a code line is preceded by a comma and an identifier, then the identifier is assigned as its value the absolute core address into which the code line will be loaded as an absolute Linc Instruction. An identifier not inherently defined is called a tag.

An expression, e, is defined to be any string of identifiers and octal constants separated by blanks, minus signs, or plus signs. An expression must be part of a single line of text, since a carriage return cannot be part of an expression. The value of an expression, \( v(e) \), is the appropriate arithmetic combination of the values of its components:

\[
\begin{array}{ll}
\text{e} & \text{v(e)} \\
20 & 0020 \\
\text{JMP} & 6000 \\
\text{JMP+20} & 6020 \\
\text{JMP 20} & 6020 \\
20\text{JMP} & 6020 \\
\text{JMP20} & ?? \\
\text{JMP-20} & 5760 \\
5-1+1 & 0005 \\
5-1 1 & 0005 \\
5-1-1 & 0003 \\
\text{STC} & 4000 \\
\text{ADD} & 2000 \\
\text{ADD STC} & 6000 \\
\end{array}
\]

Note: (1) Plus signs and spaces are completely inter-changeable. (2) No separator is necessary when a number is followed by an identifier. (3) A separator is necessary when a number follows an identifier. Lass interprets the concatenation as a new longer identifier whose value, if it is indeed defined, has no necessary relationship to the values of its constituents. (4) A minus sign negates the value of only the immediately following item. (5) No specific structure (such as INSTRUCTION/ADDRESS) is expected by Lass, rather all elements of an expression are merely evaluated as they are encountered and combined to yield a single numerical value.
A code line has one of the following structures, where \( \mu \) is any tag and \( \gamma \) is a carriage return:

\[
\text{\( e\gamma \)} \quad \text{or} \quad \text{\( \mu \, e\gamma \)}
\]

In both cases, the absolute code generated is \( v(e) \). In the second case, however, the tag \( \mu \) becomes the symbolic address of the code line.

**Comments**

Any string of characters included between parenthesis is ignored by Lass.*

<table>
<thead>
<tr>
<th>symbolic line</th>
<th>absolute code</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) JMP 20 (START OVER)</td>
<td>6020</td>
</tr>
<tr>
<td>(b) (THIS CAN MISLEAD) (THIS IS THE WAY TO HAVE VERY LONG, MULTI-LINE COMMENTS)</td>
<td>6030</td>
</tr>
<tr>
<td>(c) JMP 30</td>
<td></td>
</tr>
</tbody>
</table>

(a) shows how a line of text can easily be annotated. (b) shows that a comment on a line by itself has the same effect as a completely blank code line, and results in the absolute code 0000 (a halt). (c) shows how a comment can be on a line by itself while at the same time not producing a spurious instruction.

**Alphabetic constants**

Lass interprets characters included between quotes as BCD data to be combined to form a numeric constant:

<table>
<thead>
<tr>
<th>symbolic line</th>
<th>absolute code</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;A&quot;</td>
<td>4100</td>
</tr>
<tr>
<td>&quot;A&quot;</td>
<td>0041</td>
</tr>
<tr>
<td>&quot;AA&quot;</td>
<td>4141</td>
</tr>
<tr>
<td>&quot;BB&quot;</td>
<td>0042</td>
</tr>
<tr>
<td>&quot;BB&quot; - &quot;A&quot;</td>
<td>0100</td>
</tr>
</tbody>
</table>

*Thus line numbering may not be consistent between Edit and Lass.*
the i-bit

The semicolon, ;, is used to represent the i-bit. Thus the occurrence of a ; in an expression has the same effect as the occurrence of the constant 20.

DEFINE

Besides being used as a symbolic address, a tag may be assigned a value through the DEFINE pseudo-op:

```
DEFINE μ = e;
```

assigns the value \( v(e) \) to the identifier (tag) \( μ \). In this case, however, \( e \) must be a defined expression, i.e., every identifier occurring in \( e \) must have been assigned a value prior to the occurrence of the DEFINE command. Thus

```
JMP A+1,
,A NOP
```

is a permissible sequence, whereas

```
DEFINE B = A+1,
,A NOP
```

is not.

ORG

Associated with each code line is its location: the absolute Linc address into which it will be loaded. The location normally assumes sequential values: if \( n \) is the location of a given code line, then \( n+1 \) will be the location of the following code line (even if this next code line begins a new text). The programmer can, however, set the location to any desired value by using the ORG pseudo-op. The line

```
ORG e7
```

causes the location of the next code line to be \( v(e) \). Succeeding
code lines will then have locations $v(e)+1$, $v(e)+2$, etc., until the occurrence of another ORG. The initial ORG of a program (appearing before any code line in the first text) can have any value between 0011 and 3777; all other ORG commands must never decrease the value of the location. If not specified, the initial location is 0400.

current location

To the period, . , Lass assigns the value of the location of the code line on which the . appears:

<table>
<thead>
<tr>
<th>location</th>
<th>symbolic line</th>
<th>absolute code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0520</td>
<td>JMP.1</td>
<td>6521</td>
</tr>
<tr>
<td>0521</td>
<td>JMP.-2</td>
<td>6517</td>
</tr>
</tbody>
</table>

GLOBAL

Texts are symbolically isolated from one another, i.e., tags occurring in one text are not related to tags in other texts, even though they may be spelled identically. This feature of Lass is meant to allow subprograms to be written at different times, or by different programmers, and then combined with other subprograms by Lass without resulting in duplicate tag definitions. In order to allow communication among subprograms, however, some tags can be specified as GLOBAL; these tags can then be used in more than one text with the same meaning. The form of this meta-command is

GLOBAL $\mu_1 \mu_2 \mu_3 \cdots \mu_n$

Global commands must precede all other lines of a given text, but any number of them may appear. The subtleties of GLOBAL are demonstrated in the example on the following page.
Note: (1) Within a given text, a tag is only considered to be global if it occurs in the GLOBAL declaration of that text. (2) The code resulting from different texts is loaded into core in the same order that the texts are given as parameters to Lass.

There is another way in which tags can be made common to more than one text: if two texts are meant to be part of the same text, but perhaps their combined length exceeds the 4 block maximum set by EDIT, then Lass will regard them as the same text if the last line of the first text is CONT (continue). If the text is divided into 3 sections, then both the first and second parts must end with CONT, etc. If the example at the top of this page is modified by appending CONT to the contents of text 1, then the following conditions will arise: (1) The declaration GLOBAL A on the first line of text 2 will become an illegal operation, since it is no longer the first line of a text (as interpreted by Lass). (2) Line 3 of text 3 will be flagged as a duplicate tag definition, since the definition of B
appearing on line 4 of text 2 now comes within the range of text 1's global declaration.

DITTO

The sequence

\[ DITTO \; e_2 \]

instructs Lass to assemble the expression \( e_1 \) and to place its value into \( y(e_2)+1 \) successive locations. \( e_2 \) must be a defined expression, though \( e_1 \) may be an arbitrary code line:

, A 400 B
DITTO 3

is equivalent to

, A 400+B
400+B
400+B
400+B

DITTO is normally used to reserve a block of storage for data:

, TABLE
DITTO 77

will allow TABLE+0, TABLE+1, ..., TABLE+77 to be available as an array of 100 elements, all initially containing 0, since the code line +NY results in the absolute code 0000.

VARB

The two halves of Linc memory (locations 0-1777 are the first, or lower, half; 2000-3777 are the second, or upper, half) have significantly different properties. E.g., the full-word class Linc instructions (ADD, STC, JMP) refer to only half of memory, and instructions cannot normally be executed in the upper half.

This dichotomy normally makes it desirable to place the actual code of a program (the executable instructions) in lower core, while
allocating storage space to blocks of data in upper core.

Such a programming technique is easily implemented in Lass through use of the VARB pseudo-op. Specifically, a line containing the word VARB may occur once in a text: all lines in the text which precede VARB are defined to be in code-space, while all lines in the text which follow VARB are defined to be in varb-space. Lass assembles all lines of code-space (in the order that the texts were given as parameters) and then goes back and assembles varb-space (beginning again with the first text).

The following example demonstrates the effect of VARB on the order of location assignment to code lines.

<table>
<thead>
<tr>
<th>Text no.</th>
<th>Order of Assembly</th>
<th>Location</th>
<th>Symbolic Line</th>
<th>Abs. Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td></td>
<td>1000</td>
<td>LDA 1000</td>
<td>1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1001</td>
<td>A 2000</td>
<td>2000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1002</td>
<td>STC 20</td>
<td>4020</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>VARB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2000</td>
<td>ORG 2000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>,A 37</td>
<td>0037</td>
</tr>
<tr>
<td>4th</td>
<td></td>
<td>2001</td>
<td>DITTO 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2002</td>
<td></td>
<td>0037</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2003</td>
<td></td>
<td>0037</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2004</td>
<td></td>
<td>0037</td>
</tr>
<tr>
<td>2nd</td>
<td></td>
<td>1003</td>
<td>ORG 1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1004</td>
<td>STA 1020</td>
<td>1020</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>VARB</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2005</td>
<td>,B 0000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2006</td>
<td>,A 300 003</td>
<td></td>
</tr>
<tr>
<td>3rd</td>
<td></td>
<td>1005</td>
<td>ORG 1000</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1006</td>
<td>STA 1020</td>
<td>1020</td>
</tr>
</tbody>
</table>

33.
There is a special mode of operation under which the Linc can execute instructions in upper memory. Execution of the instruction 0010, while in lower core, causes the next JMP instruction (6000+X) to set the location counter to 2000+X (necessarily in upper core) rather than to X, as would normally occur. Once the location counter is in upper core, lower core is no longer addressable: the highest-order bit in the effective address of each instruction is assumed to be on. Thus in this mode

```
ADD 40 LDA LDA JMP 40
40 2040
```

all refer to cell 2040.

Execution can be returned to lower core, where both halves of core are addressable, by executing 0010 a second time. The next JMP instruction (6000+X) will set the location counter to X (necessarily in lower core).

The pseudo-op UPPER allows the programmer to easily write routines which can be executed in upper core:

<table>
<thead>
<tr>
<th>location</th>
<th>symbolic line</th>
<th>absolute code</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>ORG 400</td>
<td>--</td>
</tr>
<tr>
<td>0400</td>
<td>MSC 10</td>
<td>0010</td>
</tr>
<tr>
<td>0401</td>
<td>JMP A</td>
<td>6001</td>
</tr>
<tr>
<td>--</td>
<td>ORG 2001</td>
<td>--</td>
</tr>
<tr>
<td>--</td>
<td>UPPER</td>
<td>--</td>
</tr>
<tr>
<td>2001</td>
<td>,A JMP.1</td>
<td>6002</td>
</tr>
<tr>
<td>2002</td>
<td>MSC 10</td>
<td>0010</td>
</tr>
<tr>
<td>2003</td>
<td>JMP 20</td>
<td>6020</td>
</tr>
<tr>
<td>2004</td>
<td>A</td>
<td>0001</td>
</tr>
</tbody>
</table>

Thus, the effect of UPPER is to subtract 2000 from the value of the symbol , , and to reduce the value of symbolic address definitions by 2000.
Blink Programs

Blink programs are translated by the Blink compiler (on the 7090) into relocatable Linc programs. The relocatable octal code is present on the BCD output tape (unit A3), and is loaded by the BLINK program (a member of the Loss program stack) from the Datamec tape unit onto a file. Such files can be given as parameters to Lass just as texts can be. The executable instructions and most variables of Blink files are assembled into code-space, while arrays are part of varb-space. Subroutines and Procedures written in Lass (i.e., in the symbolic assembly language) can be referenced symbolically in Blink: an identifier declared to be an EXTERNAL PROCEDURE or an EXTERNAL SUBROUTINE in Blink will be recognized by Lass as a global identifier. The Lass text in which the identifier is defined (and in which the desired routine is written) must include the identifier in its global declaration. Apart from the EXTERNAL identifiers, there is no symbolic communication between Blink programs and Lass programs.

An example:

Blink program: A.. EXTERNAL SUBROUTINE COMP3 $ 13=27 $ ENTER COMP3 $ GO A $  

Lass text: GLOBAL COMP3 ,COMP3 LDA 3 COM STC 3 JMP 0  

Approximate resulting code:

<table>
<thead>
<tr>
<th>Address</th>
<th>Opcode</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0400</td>
<td>0063</td>
<td>(SET;3)</td>
</tr>
<tr>
<td>0401</td>
<td>0027</td>
<td></td>
</tr>
<tr>
<td>0402</td>
<td>6404</td>
<td>(JMP COMP3)</td>
</tr>
<tr>
<td>0403</td>
<td>6400</td>
<td>(JMP A)</td>
</tr>
<tr>
<td>0404</td>
<td>1000</td>
<td>(LDA)</td>
</tr>
<tr>
<td>0405</td>
<td>0003</td>
<td></td>
</tr>
<tr>
<td>0406</td>
<td>0017</td>
<td></td>
</tr>
<tr>
<td>0407</td>
<td>4003</td>
<td></td>
</tr>
<tr>
<td>0410</td>
<td>6000</td>
<td></td>
</tr>
</tbody>
</table>
ERRORS

If one of the following error conditions is detected by Lass, assembly is terminated, and an error message is typed describing the error and specifying the text and line number of occurrence:

1. encountering of file which is neither alphabetic nor Blink
2. encountering of improper character in a text
3. duplicate tag definitions
4. failure of a tag to be defined
5. improper use of a pseudo-operation

When such an error occurs, the programmer corrects the fault and re-calls the assembler. Some frequent sources are as follows:
Error (1) often is caused by failure of the programmer to include the desired program stack number in his call of Lass. Error (2) could result either from a typographical error or missing parenthesis in a comment. Error (3) could easily occur in a long text. Error (4) could result from an oversight during the creation of a text, or from failure to include a required text in the call of Lass. Error (5) could result from an ORG which tries to decrease the program counter or a DEFINE or DITTO which employs an undefined expression. The descriptive error message, together with such features of the EDIT program as _F_ (Find), make the correction of most of these syntactical errors straightforward and fast.

Symbol-Table Printout

If sense switch 5 is on during assembly, Lass will print a list of all tags encountered together with their values. Such a symbol table, together with listings of the individual texts, makes the problem of debugging the running program fairly straightforward. A symbol table for Blink programs (with addresses relative to the loading address of the Blink program, which is almost always 0400) is included in the BCD output of each compilation.
Since index registers are often used in Linc programs to reference sequential memory locations, and due to the division of core into upper and lower halves, care should be taken that strings, arrays, or other multi-word items do not span the core interface. An ORG 2000 immediately preceding varb-space will of course avoid such a difficulty. In the case of self-sufficient Blink program (i.e., one with no external procedures or subroutines) which is large enough that part of its varb-space will extend beyond location 2000, a single Lass text containing just the line ORG 2000, assembled together with the Blink file, will force all of the Blink program's varb-space into upper core.

**Program Stack Updating**

A Lass assembly might merely create an updated version of an already existing program; in this case the program name will already be present in the monitor's program stack index. In case an entirely new program is created, however, the name must be added to the index as follows: The index exists somewhere on unit 1 as a text. The appropriate change is made to the text by use of EDIT (The exact format of program name entries will be evident upon examination of this text). The updated text is made available to the monitor by transferring the text (limited to a length of one block) from its position on unit 1 to block 0155 of unit 0 (the 5th block of the monitor program). This tape transfer is accomplished by halting the computer and performing tape operations through the switches.

In addition, any program stack location can be called from the monitor by typing the location number rather than a program name. In this case the parameter request message is always the non-descript PARAMS>>. Thus programs of only temporary usefulness can be assembled and used without updating the program index.
### Inherently Defined Lass Identifiers

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Identifier</th>
<th>Value</th>
<th>Identifier</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLT</td>
<td>0000</td>
<td>OPR</td>
<td>0500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLR</td>
<td>0011</td>
<td>SNS</td>
<td>0440</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MSC</td>
<td>0000</td>
<td>AZE</td>
<td>0450</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ATR</td>
<td>0014</td>
<td>APO</td>
<td>0451</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTA</td>
<td>0015</td>
<td>LZE</td>
<td>0452</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>0016</td>
<td>IBZ</td>
<td>0453</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>0017</td>
<td>SXL</td>
<td>0400</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROL</td>
<td>0240</td>
<td>KST</td>
<td>0415</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ROR</td>
<td>0300</td>
<td>RDC</td>
<td>0700</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SCR</td>
<td>0340</td>
<td>RCG</td>
<td>0701</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADD</td>
<td>2000</td>
<td>RDE</td>
<td>0702</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STC</td>
<td>4000</td>
<td>MBT</td>
<td>0703</td>
<td></td>
<td></td>
</tr>
<tr>
<td>JMP</td>
<td>6000</td>
<td>WRC</td>
<td>0704</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>1000</td>
<td>WCG</td>
<td>0705</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STA</td>
<td>1040</td>
<td>WRI</td>
<td>0706</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADA</td>
<td>1100</td>
<td>CHK</td>
<td>0707</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADM</td>
<td>1140</td>
<td>SAVE</td>
<td>0020</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAM</td>
<td>1200</td>
<td>RESTORE</td>
<td>0046</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUL</td>
<td>1240</td>
<td>PARAMS</td>
<td>0074</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAE</td>
<td>1440</td>
<td>REPEAT</td>
<td>0123</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRO</td>
<td>1500</td>
<td>STEP</td>
<td>0146</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCL</td>
<td>1540</td>
<td>OVERLAY</td>
<td>0343</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCO</td>
<td>1640</td>
<td>POINTER</td>
<td>0155</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BSE</td>
<td>1600</td>
<td>PUT</td>
<td>0164</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DSC</td>
<td>1740</td>
<td>GET</td>
<td>0214</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LDH</td>
<td>1300</td>
<td>GETCL</td>
<td>0235</td>
<td></td>
<td></td>
</tr>
<tr>
<td>STF</td>
<td>1340</td>
<td>LSTOP</td>
<td>0226</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHD</td>
<td>1400</td>
<td>LSTCL</td>
<td>0241</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td>0040</td>
<td>STRING</td>
<td>0244</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SAM</td>
<td>0100</td>
<td>LSTEL</td>
<td>0263</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIS</td>
<td>0140</td>
<td>RETURN</td>
<td>0336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XSK</td>
<td>0200</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*See Loss appendix for explanation of some of these identifiers.*
Quarters 0 and 6 of Linc memory are reserved by Loss for certain utility routines and an inter-program communication buffer. This description of the use of these routines, and of their relation to the total Loss operational structure, constitutes an introduction to both the Blink write-up and the Loss appendix.

the overlay stack

Certain positions on the program stack (locations 17 through 22) are reserved for a recursive overlay stack; these locations are not available for use as part of the regular program stack.

OVERLAY and RETURN

OVERLAY loads a program from the program stack into core, saving the previous contents of memory on the overlay stack. OVERLAY is called by placing the number of the desired program stack location in the accumulator and executing the instruction JMP OVERLAY (JMP 343). OVERLAY then writes quarters 1 through 5 of Linc memory onto the corresponding 5 blocks of the first available location on the overlay stack, reads blocks 1-5 of the requested stack location into core, and starts at 400. When the "called" program has completed its execution, the instruction JMP RETURN (JMP 336) is executed. Blocks 1-5 of the appropriate location on the overlay stack are read back into core and execution resumes one location beyond the previously mentioned call of OVERLAY. Programs can thus be called just like subroutines.

The overlay stack is a recursive push-down stack; thus a program which was loaded by OVERLAY can itself call OVERLAY to execute another program.

As an example, let us assume that the monitor is used to call
a program named TYPE which itself calls a program named QA (Question and Answer) which accepts the name of a file to be typed by TYPE. Further assume that TYPE occupies stack location 3, and that QA occupies stack location 4. Under these assumptions, when the user supplies the name TYPE to the monitor, together with an empty parameter list, the monitor will place 0003 in the accumulator and execute JMP OVERLAY. Quarters 1-5 will be placed on tape blocks 171-175, blocks 31-35 will be loaded into core, and execution will begin at 400. The loaded TYPE program will then place 0004 in the accumulator and JMP to 0343. Quarters 1-5 will be written on blocks 201-205, blocks 41-45 will be read into core, and control will transfer to 400. When QA has accepted the appropriate parameters, and has executed JMP RETURN, blocks 201-205 will be read back into core and execution of TYPE will resume where it left off. TYPE may call QA many times to get further parameters. When TYPE has finally finished, it does a JMP RETURN: blocks 171-175 (the saved monitor) are read back into core, and the monitor requests another program call.

the buffer

The buffer, which occupies quarter 6, is a structured data storage area. The unit of data storage is the record: any number of records can occupy the buffer at the same time, and each record consists of any number of variables or strings. The buffer is a push-down stack, i.e., the last record added to the buffer is always the first to be removed. The buffer can be used to manipulate data lists within a single program, however since quarter 6 is left constant by the overlay routine, the most common use of the buffer is to transmit data between different programs. In particular, parameters accepted by
the monitor are merely placed on the buffer in the form of a single record. When a program is then loaded by the monitor, the program extracts its parameters from the top record on the buffer and then erases that record. Thus any program which can be called by the monitor can be called by any other program: so long as the appropriate record is present on the top of the buffer, a program can perform its function independent of the program which called it (as a matter of fact, a program can recursively call itself).

PUT and GET

PUT and GET are procedures (located in quarter 0) which can either put a record onto the buffer, or get a record from the buffer. The parameter of each of these procedures is a list of variables or strings. The structure of such lists is explained in the appendix, but an understanding of their structure is not necessary in this description. PUT examines each item in the referenced list and places the appropriate value in a new record on the buffer. The value of a variable is its numeric value, while the value of a string is an entire array of numeric values, usually interpreted as BCD information. As each item is placed on the buffer, it is preceded by a control word which describes the type of the item (variable or string) and tells how many words it occupies. GET extracts values from the top record of the buffer, stores them into the variables and strings present in the list, and then erases the record on the buffer.

REPEAT

REPEAT is a routine which can be used to govern the execution of iterative loops. The efficiency of a REPEAT-governed loop is approximately the same as the conventional Lass loop which uses the XSK instruction. Thus while REPEAT offers no advantage to the Lass
programmer, the efficiency REPEAT contributes to Blink programs is considerable.

**PARAMS**

PARAMS is a procedure which facilitates the transfer of parameters to other procedures. Its use is automatic for all procedures compiled by Blink, and the use of PARAMS in Lass programming is explained in the appendix.

The routines in quarter 0 place some restrictions upon the use of index registers. Index registers 11-17 are used by these routines, but may also be used by Lass programs, though the quarter 0 routines do not restore the contents of these registers to the values held before entry to quarter 0. Index register 10 may not be used by Lass programs at all; its integrity is required for proper execution of the Repeat routine. Registers 1-7 are never affected by quarter 0.

The Lass programmer need not learn the full details of most of the routines in quarter 0, since they are of use mainly to Blink programs. However, a knowledge of the structure of the buffer is required to access parameters accepted by the monitor; an understanding of the use of OVERLAY and RETURN is necessary in order to control the loading of programs and the return to monitor; and a knowledge of PARAMS is required so that EXTERNAL PROCEDURE's can be written in Lass in such a way that they can be called by Blink programs.

**Loss conformability**

If it is desired to use a program under Loss which does not respect the inviolability of quarter 0 or the buffer (e.g., a program written without Loss in mind), the following procedure may be used:

1. Fit the program onto a program stack location.
2. Expect quarter 1 of the program to be loaded and for execution to begin at 400. Other program loading can be done by the program itself in a bootstrap fashion.
(3) Return to monitor by executing the sequence:

```
RDC
0150
JMP 0
```

which reloads the virgin monitor.

Alternatively, of course, quarters 0 and 6 can be written on tape during program execution. Return to monitor would then be accomplished by:

```
RDC
0???
RDC
6???
JMP RETURN
```

**buffer dump**

Whenever control is returned to the monitor from another program, the monitor types out the contents of the buffer--erasing all buffer records--before requesting the next program call. This feature makes it very easy for programs to transmit a message to the user upon completion of their execution.
BLINK

General Description

BLINK is a version of Subalgol designed for use with the LINC computer. Programs very similar to Subalgol programs are translated on the 7090 by the BLINK compiler (which is written in Subalgol), into relocatable LINC code.

Reserved Word Changes with Semantics

BLINK has no "library procedures", though it retains all of Subalgol's "intrinsic functions". The following Subalgol reserved words are without special meaning in BLINK:

STOP, SHLT, SHRT, EXTR, STATEMENT, WHILE, SEGMENT, MONITOR, STEP,
INPUT, OUTPUT, TRACE, DPrecision, LIBRARY, CARDREAD, PRINTOUT,
COMPLEX, RE, IM, WRITE, READ, SQRT, LOG, EXP, SIN, COS, TAN,
ENTIRE, SINH, COSH, TANH, ARCTAN, ROMXX, ARCSIN, ARCCOS, RCARD,
READM, WRITEM, CHECKM, MOVEM, MOVEFILE, ENDFILE, REWIND, UNLOAD,
FLAGM, etc.

The following reserved words are introduced or redefined with BLINK:

I.

A. ROTL, ROTR, SCLR
B. BTCLR, BTCOM, BTSET
C. LDA
D. STA
E. DO
F. REPEAT
G. RDC, RCG, MTB, WRC,
   WCG, WRI, CHK, RDE
H. INCR.
I. OVERLAY
J. STRING
K. LIST
L. PUT, GET
M. *QUIT
N. *GETCOR
O. EXTERNAL
P. SNS
Q. EXIT
R. RETURN
II.
A. I1, I2, ..., I7
B. M, MH
C. III, I2I, ..., I7I
D. POINTER

**Corresponding Semantics**

I.
A. ROTL(N,OPERAND) : Intrinsic function; arguments type integer; result type integer; corresponds to ROL instruction; as in later intrinsic functions, the effect of the 1-bit is obtained by using a value of N > 17.
B. BTCLR(MASK,OPERAND) : Intrinsic function; types integer; corresponds to BCL, etc.; as with ROTL class function, a constant first argument naturally reduces length of resulting code.
C. LDA(<arbitrary arithmetic expression>) : Expression is calculated and placed in accumulator; useful in connection with DO and STA as mentioned below.
D. STA(<simple variable>) : Contents of the accumulator are placed in the simple variable.
E. DO(< arithmetic expression>) : Expression is evaluated, treated as a LINC instruction and executed, e.g.,*

```
LDA(I)$
DO("470")$ COMMENT AZE1$ GO TO L$
STA(J)$
```

* Within BLINK examples, numbers are decimal unless placed in double quotes.
is identical in effect to:

EITHER IF I EQL 0$ GO TO L$

OTHERWISE$ J=I$

The DO function, however, is of only dubious value as a tool to create tight code; its real purpose is to allow the use of external device communication instructions in BLINK programs, e.g., OPR, SXL, etc.

F. REPEAT(<integer expression>)$ <statement>$ : Identical in effect to:

DMY1=<integer expression>$

FOR DMY2=(1,1,DMY1)$ <statement>$

except that the REPEAT loop is more efficient and does not change the value of any variable in its indexing.

G. RDC(i,u,QNMBR,BNMBR) : Identical to LASS$ except that i and u are represented by a 0 or 1.

H. INCR(<expression>,<variable>) : Identical in effect to:

<variable>=<variable>+<expression>

except that if the variable is subscripted, INCR calculates the subscript only once and INCR is a function having the new value of <variable> as its value.

I. OVERLAY<integer expression> : The OVERLAY* routine is entered with the integer expression in the accumulator.

J. STRING<identifier>(<integer>)=(<alpha string>) : The STRING declaration is identical to the ARRAY declaration, except that only a single dimension, and no irregular subscript ranges, are allowed. The effect of the STRING declaration is different in

* The reader should be familiar with LOSS at this point.
that the zero'th position of the STRING (even though not requested in the declaration) is reserved and filled with the size of the STRING, this information being necessary to the PUT and GET routines. STRINGS may be manipulated word by word, as are ARRAYS, through subscription. Thus S(1) refers to the first and second characters of the STRING S.

K. LIST: The LIST declaration is identical to the Subalgol OUTPUT declaration except that a STRING name (followed by empty parenthesis) is allowed as a LIST element, and fulfills the role served by the alphanumeric insertion phrase in Subalgol. A LIST, however, is somewhat more elegant than a Subalgol INPUT or OUTPUT list since a LIST can be used for either input or output (i.e., as argument of either GET or PUT), and includes the types of its elements, therefore needing no accompanying FORMAT (which concept therefore fails to exist in BLINK).

L. PUT,GET: These are simply procedures (always in core) which can have any number of LISTS as program reference parameters.

M. *QUIT: This control card should follow the FINISH card of the last of any group of BLINK programs. It causes the BLINK compiler to write an end of file on the output tape and returns control to the 7090 monitor.

N. *GETCOR: BLINK compilation can also be terminated by loading another disc program, e.g.

*GETCOR 225 CRDTOTAP
could conceivably cause data cards to be tacked onto the end of a tape containing BLINK compilations.
O. EXTERNAL PROCEDURE, EXTERNAL SUBROUTINE: These declarations, identical to those in Subalgol, allow linkages to be created on the LINC between BLINK subprograms and subprograms created by means other than the BLINK compiler.

P. SNS n: SNS 0, SNS 1, ..., SNS 5 are predicates which are true when the corresponding sense switches are on.

Q. EXIT: Repeat loops may be recursively nested, but the nesting must be perfect; thus if a repeat loop is to be irregularly terminated, departure must be by means of EXIT rather than GO TO. E.G.:

```
REPEAT N $
BEGIN
GO TEST $
L.. IF ERROR $ EXIT NOGOOD $
END $
```

where 'TEST' and 'NOGOOD' are both labels, but it is assumed that 'TEST' always returns to 'L' whereas transfer to 'NOGOOD' terminates the loop.

R. RETURN: This statement, if it occurs outside of a subroutine or procedure declaration, compiles into the instruction JMP RETURN and means: "Return to the program which Overlayed this program".
II.

A. Unlike Subalgol, BLINK has reserved variables. I1 through I7 (index registers), are simple variables of type integer with absolute address 1 through 7, respectively. These variables are GLOBAL and their values are not restored after an overlay.

B. M and MH are GLOBAL arrays with absolute base address of 0. Their types are integer and half-word, respectively. These are used to great advantage together with I1 through I7:

\[ M(I1) = M(I2) \] results in the elegant code

LDA 2, STA 1

C. I11, ..., I17 are used in conjunction with the M() and MH() arrays in order to reference consecutive words, or half-words, of core.

As an example, the following statements replace the contents of quarter 5 by the contents of quarter 4:

\[ I1 = "3777" , I2 = "2377" \]

REPEAT "400" $ M(I11) = M(I11) $

COMMENT LDA11, STA12 $

Thus the value of the indicated index register is incremented before it is used as a subscript. When the above program is completed, I1 will contain "2377" and I2 will contain "2777". (It is a quirk of the LINC that the core is logically divided into halves; thus 3777 is the predecessor of 2000 and 1777 is the predecessor of 0.)

49.
In the case of half-words, index registers are incremented by 4000 rather than by 1. Thus if an index register were stepping through the characters of quarter 4, it would assume successively the values 2000, 6000, 2001, 6001, 2002, etc. The 4000-bit indicates the right-half of the word.

An index register can be made to point at a variable by a statement of the form \( \text{InI} = \text{variable} \). The code generated is:

\[
\text{SET in} \\
\text{<variable location>}
\]

The following program places the characters of the STRING \( S() \) into quarter 7, putting one character (right justified) into each word.

\[
\begin{align*}
\text{STRING } S(20) = & \text{ (alpha string )} \\
I7I = & "3377" \\
\text{COMMENT: There is canonical correspondence between registers and quarters} \\
I2I = & S(0) \ I2 = I2 + "4000" \\
\text{REPEAT 40} & M(I7I) = MH(I2I) \\
\text{COMMENT: LDH} & i2, \text{ STA} i7
\end{align*}
\]

D. \text{POINTER} is that cell ("155") in QUARTER 0 which points to the top of the BUFFER. The statement \( \text{POINTER} = M(\text{POINTER}) \) would erase one record from the BUFFER. If we assume that the top record of the BUFFER begins with an alphabetic item, then the statement:

\[
I2 = M(\text{POINTER}) + "4001"
\]

would allow \( MH(121) \) to reference successive characters of that first item.
Since the Balgo procedures READ and WRITE do not exist in Blink, input-output requires creative programming on the part of the Blink user. The following examples exhibit solutions to the problem of typewriter input-output as well as providing examples of complete Blink programs.

Each of these programs is a primitive "desk-calculator": given two octal numbers from the teletype, our programs will print out the arithmetic product.

**method 1:** avoiding external procedures:

```
STRING REQUEST(5)=('FACTORS..'),
ANSR(6)=('PRODUCT IS..')
LIST FACTORS (X,Y), PRODUCT (ANSR(),X,Y),
ASK(REQUEST())
GET($$FACTORS)$ COMMENT ERASE PARAMETERS SUPPLIED BY MONITOR$
PUT($$ASK)$ COMMENT MESSAGE TO BE TYPED BY QA$
NEXTVALS.. OVERLAY 23$ COMMENT QA PROGRAM$
GET($$FACTORS)$ COMMENT INPUT OF X AND Y$
IF X EQL 0$ RETURN$ COMMENT TERMINATION CONDITION$
PUT($$PRODUCT,ASK)$ COMMENT BOTH LISTS COMBINE INTO ONE RECORD$
go NEXTVALS$
FINISH$
```

**method 2:** avoiding tape shuffling:

```
(same list declarations)
GET($$FACTORS)$ COMMENT AGAIN MERELY ERASE BUFFER$
PUT ($$ASK)$ COMMENT MESSAGE TO BE TYPED.$
EXTERNAL SUBROUTINE TYPEREC $ COMMENT THIS ROUTINE TYPES A RECORD$
EXTERNAL SUBROUTINE INREC $ COMMENT THIS ROUTINE ACCEPTS A RECORD$
NEXTVALS.. ENTER TYPEREC $ COMMENT TYPE PROPER MESSAGE$
ENTER INREC $ COMMENT ACCEPT DATA FROM TELETYPE$
GET ($$FACTORS )$
IF X EQL 0$ RETURN$ PUT($$PRODUCT,ASK)$
go NEXTVALS$
FINISH$
```
By both of the above methods, the user-Linc communication could look about like this:

```
<PROG>>
* MULTIPLY
NIL>>
* ;
<RUN>>
FACTORS..>>
* -2
* 3
* ;
<RUN>>
PRODUCT IS..
7771
FACTORS..>>
* 2
* 4
* ;
<RUN>>
PRODUCT IS..
10
FACTORS..>>
* 0
* ;
<RUN>>

<PROG>> (monitor reloaded)
```

As a final example, the following program finds the product of only a single pair of numbers:

```
(same list declarations)
GET($FACTORS)$ COMMENT USE MONITOR FOR INPUT $
PUT($PRODUCT)$ COMMENT MONITOR AUTOMATICALLY DUMPS BUFFER $
FINISH $ COMMENT RETURN AUTOMATICALLY COMPILED AT END OF PROGRAM $
```

In this case the on-line listing could look like this:

```
<PROG>>
* MULTIPLY
FACTORS>>
* 2
* 3
* ;
<RUN>>
PRODUCT IS..
6
<PROG>>
```
<table>
<thead>
<tr>
<th>ADDRESS</th>
<th>INSTRUCTIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>a000</td>
<td>16</td>
</tr>
<tr>
<td>a001</td>
<td>CLR</td>
</tr>
<tr>
<td>a002</td>
<td>STA</td>
</tr>
<tr>
<td>a003</td>
<td>31A0</td>
</tr>
<tr>
<td>a004</td>
<td>STA</td>
</tr>
<tr>
<td>a005</td>
<td>3A00</td>
</tr>
<tr>
<td>a006</td>
<td>LCS</td>
</tr>
<tr>
<td>a007</td>
<td>451</td>
</tr>
<tr>
<td>a008</td>
<td>JMP</td>
</tr>
<tr>
<td>a009</td>
<td>HLT</td>
</tr>
<tr>
<td>a010</td>
<td>HLT</td>
</tr>
<tr>
<td>a011</td>
<td>HLT</td>
</tr>
<tr>
<td>a012</td>
<td>HLT</td>
</tr>
<tr>
<td>a013</td>
<td>HLT</td>
</tr>
<tr>
<td>a014</td>
<td>HLT</td>
</tr>
<tr>
<td>a015</td>
<td>HLT</td>
</tr>
<tr>
<td>a016</td>
<td>HLT</td>
</tr>
<tr>
<td>a017</td>
<td>HLT</td>
</tr>
<tr>
<td>a018</td>
<td>SET</td>
</tr>
<tr>
<td>a019</td>
<td>LDA</td>
</tr>
<tr>
<td>a020</td>
<td>1776</td>
</tr>
<tr>
<td>a021</td>
<td>REPEAT</td>
</tr>
<tr>
<td>a022</td>
<td>ADD</td>
</tr>
<tr>
<td>a023</td>
<td>STA</td>
</tr>
<tr>
<td>a024</td>
<td>LDA</td>
</tr>
<tr>
<td>a025</td>
<td>15</td>
</tr>
<tr>
<td>a026</td>
<td>LDA</td>
</tr>
<tr>
<td>a027</td>
<td>LDA</td>
</tr>
<tr>
<td>a028</td>
<td>ROL</td>
</tr>
<tr>
<td>a029</td>
<td>SCR</td>
</tr>
<tr>
<td>a030</td>
<td>LDA</td>
</tr>
<tr>
<td>a031</td>
<td>STA</td>
</tr>
<tr>
<td>a032</td>
<td>LDA</td>
</tr>
<tr>
<td>a033</td>
<td>LDA</td>
</tr>
<tr>
<td>a034</td>
<td>STA</td>
</tr>
<tr>
<td>a035</td>
<td>LZE</td>
</tr>
<tr>
<td>a036</td>
<td>JMP</td>
</tr>
<tr>
<td>a037</td>
<td>LDA</td>
</tr>
<tr>
<td>a038</td>
<td>21</td>
</tr>
<tr>
<td>a039</td>
<td>STA</td>
</tr>
<tr>
<td>a040</td>
<td>LDA</td>
</tr>
<tr>
<td>a041</td>
<td>15</td>
</tr>
<tr>
<td>a042</td>
<td>LDA</td>
</tr>
<tr>
<td>a043</td>
<td>?</td>
</tr>
<tr>
<td>a044</td>
<td>LDA</td>
</tr>
<tr>
<td>a045</td>
<td>JMP</td>
</tr>
<tr>
<td>a046</td>
<td>LDA</td>
</tr>
<tr>
<td>a047</td>
<td>36</td>
</tr>
<tr>
<td>a048</td>
<td>LDA</td>
</tr>
<tr>
<td>a049</td>
<td>21</td>
</tr>
<tr>
<td>a050</td>
<td>STG</td>
</tr>
<tr>
<td>a051</td>
<td>SET</td>
</tr>
<tr>
<td>a052</td>
<td>21</td>
</tr>
<tr>
<td>a053</td>
<td>LDA</td>
</tr>
<tr>
<td>a054</td>
<td>15</td>
</tr>
<tr>
<td>a055</td>
<td>AZF</td>
</tr>
<tr>
<td>a056</td>
<td>HLT</td>
</tr>
<tr>
<td>a057</td>
<td>STA</td>
</tr>
<tr>
<td>a058</td>
<td>15</td>
</tr>
<tr>
<td>a059</td>
<td>LDA</td>
</tr>
<tr>
<td>a060</td>
<td>ROL</td>
</tr>
<tr>
<td>a061</td>
<td>SCR</td>
</tr>
<tr>
<td>a062</td>
<td>LDA</td>
</tr>
<tr>
<td>a063</td>
<td>LDA</td>
</tr>
<tr>
<td>a064</td>
<td>ROL</td>
</tr>
<tr>
<td>a065</td>
<td>SCR</td>
</tr>
<tr>
<td>a066</td>
<td>LDA</td>
</tr>
<tr>
<td>a067</td>
<td>LDA</td>
</tr>
<tr>
<td>a068</td>
<td>STA</td>
</tr>
<tr>
<td>a069</td>
<td>LZE</td>
</tr>
<tr>
<td>a070</td>
<td>JMP</td>
</tr>
<tr>
<td>a071</td>
<td>LDA</td>
</tr>
<tr>
<td>a072</td>
<td>JMP</td>
</tr>
<tr>
<td>a073</td>
<td>JMP</td>
</tr>
<tr>
<td>a074</td>
<td>ADD</td>
</tr>
<tr>
<td>a075</td>
<td>STG</td>
</tr>
<tr>
<td>a076</td>
<td>ADD</td>
</tr>
<tr>
<td>a077</td>
<td>ADD</td>
</tr>
</tbody>
</table>

**Note:** Indicates data location

**QUARTER O**

**1/1/**
SAVE and RESTORE

These routines govern a push-down stack whose presence allows the other routines of quarter 0 to be recursive.

JMP SAVE
location_1
location_2
·
·
·
location_{n+4000}

causes the n locations together with their contents to be saved on the top of the push-down stack. The call JMP RESTORE causes the topmost list of locations on the stack to be restored to their former contents. This push-down stack occupies cells 3000 to 3140 and is called the save-buffer, as opposed to the put-buffer which begins at 3140.

PARAMS

Consider the procedure call P(3, X $ Y $ L ). The corresponding symbolic code is

LDA X
STC 3
JMP P
3
O (VALUE OF X)
Y (ADDRESS OF Y)
JMP L

All parameters are stored in memory immediately following the JMP to the procedure. Value parameters are represented by a numerical value, name parameters are represented by an absolute address, and program reference parameters are represented by an appropriate JMP
instruction. The JMP to the procedure, together with the parameters, is called the calling sequence. Upon completion of execution, the procedure normally JMP's to one location beyond the calling sequence from where it was currently called.

The PARANS routine can be used by the programmer (and is automatically used by Blink programs) to facilitate the referencing of parameters within the body of a procedure. PARANS is itself a procedure with one value parameter. The left half-word of this parameter is \( 76 - N_v \), and the right half-word is \( N_p \), where \( N_v \) is the number of value parameters expected by the procedure, and \( N_p \) is the total number of parameters. PARANS stores the return address and the value parameters in the body of the procedure (as shown below), and leaves index register 17 pointing to the last value parameter.

Consider the following Blink procedure declaration:

```plaintext
PROCEDURE P(X,Y,Z,L)
BEGIN
    Z=X+Y
    IF Z EQL 5 $ GO TO L $
END P()
```

which is equivalent to:

```plaintext
,P LDA 0
JMP PARANS (JMP 74)
7404 (76-2,4)
,RET
,X
,Y
LDA;17
STC ZADDR1
LDA 17
STC ZADDR2
LDA;17
STC JL
ADD X
ADD Y
STA
,ZADDR1 (ADDRESS OF Z)
```

LDA
,ZADDR2 (ADDR. OF Z)
ADA;
-5
AZE;
,JL (JMP L)
JMP RET

55.
The above symbolic code is meant to represent the kind of code automatically generated by the Blink compiler; the Lass programmer, however, can make somewhat more efficient use of the PARAMS routine, since he can take into account the individual characteristics of a given procedure. Here is a more optimized version of P():

```
P LDA 0
JMP PARAMS
7204 (Let all parameters be picked up by PARAMS)
,RET
,X
,ZA
,JL
SET 17
ZA
LDA X
ADD Y
STA 17
SAE;
5
JMP RET
JMP JL
```

which is 5 cells shorter than the compiled code.

REPEAT

the program

```
JMP REPEAT
JMP PAST
5

(code)

JMP STEP
,PAST (etc.)
```

causes 'code' to be executed 5 times. REPEAT is completely recursive (i.e., many REPEAT loops may be nested), and is the sole user of index register 10. A REPEAT loop can be terminated only by completing
the full number of iterations (when control will transfer to the location PAST), or alternatively, by executing the instruction JMP RESTORE followed by a JMP to any desired location outside the loop. If the count parameter of a REPEAT loop (5 in the above example) is zero or negative, the loop is not executed at all.
PUT and GET

The following program will replace each item in the list L2 by the corresponding item in the list L1. A, B, X, Y, are variables and S1() and S2() are strings.

```
STRING S1(5)= ('ALPHAS'), S2(5) $
LIST L1 (A, B, S1()) $
LIST L2 (X, Y, S2())$
PUT($$L1) $
GET($$L2) $
(etc.)
```

The corresponding symbolic code:

```
JMP PUT
JMP L1
JMP GET
JMP L2
JMP GETCL
(etc.)
,,L1 ADD 0
JMP LSTOP (BEGIN LIST)
LDA; A (ADDRESS OF A TO ACCUMULATOR)
JMP LSTEL (SEND AN ITEM TO PUT OR GET)
3776 (A CONTROL WORD; THE FIRST DIGIT IS)
LDA; (THE TYPE, 3 INTEGER, 7 IS ALPHABETIC)
B (THE FINAL 3 DIGITS ARE THE COMPLEMENT OF)
JMP LSTEL (THE SIZE OF THE ITEM)
3776
JMP STRING (SEND A STRING )
S1 (STRING ADDRESS)
JMP LSTCL (FLAG END OF LIST)
,,L2 ADD 0
(etc.)
JMP LSTCL
,S1 5 (STRING SIZE)
"AL"
"PH"
"AS"
7474 (END CODE)
0 (5th CELL NOT CURRENTLY USED)
7474 (EXTRA END CODE ADDED BY COMPILER)
,S2 5
0
DITTO 4
7474
```

58.
The BUFFER:

The BUFFER has a linked-list structure, the top of which is POINTER (cell 155). If in the previous example we assume that A and B have the values 7 and 24, respectively, then after the statement \texttt{PUT(;;L1)}, the BUFFER would have the following appearance:

<table>
<thead>
<tr>
<th>LOCATION</th>
<th>CONTENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0155</td>
<td>3154</td>
</tr>
<tr>
<td>3140</td>
<td>0000</td>
</tr>
<tr>
<td>3141</td>
<td>3776</td>
</tr>
<tr>
<td>3142</td>
<td>0007</td>
</tr>
<tr>
<td>3143</td>
<td>3776</td>
</tr>
<tr>
<td>3144</td>
<td>0024</td>
</tr>
<tr>
<td>3145</td>
<td>7771</td>
</tr>
<tr>
<td>3146</td>
<td>4154</td>
</tr>
<tr>
<td>3147</td>
<td>6050</td>
</tr>
<tr>
<td>3150</td>
<td>4163</td>
</tr>
<tr>
<td>3151</td>
<td>7474</td>
</tr>
<tr>
<td>3152</td>
<td>0000</td>
</tr>
<tr>
<td>3153</td>
<td>7474</td>
</tr>
<tr>
<td>3154</td>
<td>3140</td>
</tr>
</tbody>
</table>

If GET is ever entered when the BUFFER is empty, i.e., when location 155 contains 3140, then a halt occurs at location 223; if RESTORE is called when the SAVE-BUFFER is empty, a halt occurs at location 55.

OVERLAY:

When it is desired to OVERLAY a program from the stack, the program number is loaded into the accumulator, and JMP OVERLAY is executed. When a program has completed its function and wishes to return to its caller, JMP RETURN is executed. OVERLAY 0 is equivalent to RETURN. The sequence

\texttt{(case 1) LDAi 0005 JMP OVERLAY JMP RETURN}

is much more efficiently accomplished by:

59.
(case 2)  LDA;
    -5
    JMP OVERLAY

When OVERLAY is given a negative parameter, the contents of core
are not written out on tape before the indicated program is loaded.
When the program completes its execution, the return is one level
lower. Thus if program A calls program B (with a positive parameter)
and program B calls program C with a negative parameter, then when
program C executes JMP RETURN, control will be returned to program
A just beyond the location from where it called B.

files

Each file is represented by an entry in its book's index. The
first half word of each such entry contains the number of characters
in the file name, the next n half words contain the name itself,
and the next two half words contain the length (in blocks) of the
file and the first block occupied by the text (mod 100). Entries
are in alphabetical order and the end of the index is flagged by a
zero where the next character count would be.

Some programs (e.g. EDIT, LASS, TYPE) utilize the following
convention: The first word of a file can be used for a control word.
The left half of the word contains the current length of the file
(must not exceed the maximum length as per the index entry) and
the right half of the word denotes the kind of information currently
stored in the file. "A" (41) is alphabetic, and "B" is a Blink
compilation.
How to Run a BLINK Program.

The BLINK3 compiler is stored in the disc files of Stanford's 7090 computer. In order to use this compiler, it is necessary to prepare a card deck as follows:

<table>
<thead>
<tr>
<th>No. 1 Card:</th>
<th>SYSTEM</th>
<th>: F-INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TAPES</td>
<td>: Mount on A3, at low density, a tape which can be removed from the Computation Center.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. 2 Card:</th>
<th>SYSTEM</th>
<th>: F-INFO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Control Card:</td>
<td>Cols. 1-6</td>
</tr>
<tr>
<td></td>
<td>Cols. 7-11</td>
<td>: BLINK</td>
</tr>
</tbody>
</table>

There should follow BLINK source decks. Each BLINK program should be terminated by a FINISH CARD. Control is removed from the BLINK compiler by either a *QUIT or a *GETCOR control card.

The output produced by BLINK3 will be on the tape which was mounted on unit A3. This output is quite similar to that produced by the SUBALGOL compiler, i.e., listing of the source decks, diagnostic messages, symbol tables of the compiled programs (these being especially useful for console debugging). In addition, however, the tape will contain the actual LINC code produced by the compiler.

If no compiler error messages are produced, the tape is brought over to the LINC, mounted on the LINC's tape unit, and read by an appropriate LINC program. One such program, BLINK, merely searches the tape, ignoring all that it sees, until it comes to compiled code. That code is then transferred to the LINC tape, unit 1, in the form of a TEXT of type B.

61.
APPENDIX B
<PROG>

? .
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.63.
LASS
STACK #, FILES>

11
/2
BLINK
/0
INREC
CONVIBL
CONVRT
;
<RUN>>

CORE USED: 400 1673

<TXT 1
1174 INREC
1526 CONVRT

<TXT 2
1460 REWIND
1200 RET
1201 REC
1202 MINL
1203 MAXV
1204 PERR
1264 GOBKWD
1233 GOFWD
1303 FWD
1362 AREC
1337 APAR
1322 STOP
1250 RPAR
1246 CNCH
1346 LPAR
1310 BKWD
1314 BTH
1347 APRET
1350 APWT
1344 LPF
1443 ARET
1367 ARST
1446 HID
1373 LOD
1440 NOMO
1525 RWRT
1501 TOP
1513 TIM

BLINK
65.
LASS
STACK #, FILES>>

12
/0
DISTAPE
CONVTBL
VARBORG
INREC
BUILDIMAGE
IMAGE
DISPLAY
POTS
LASTCELL
;
<RUN>>

CORE USED: 400 2332

<TXT 1
1000 INREC
1264 REWIND
1332 NEWIMAGE
1355 ADDC
1470 ADDN
1543 ADDS
1566 DISPLAY
1643 POTS
2332 LASTCELL
2000 IMAGE
1733 CONVTBL
6000 J
401 READ
770 POT
407 E1
414 REC
411 MOV
420 NEWIM
761 ERR
735 HEDR
424 CHOOSE
637 BCD
600 SENSE
473 GRP
451 BIL
467 LINE
454 EM1
615 TEST2
716 LOWL
470 NXTL
523 GRP1
502 NHW
527 CHAR

507 CD
626 ADDB
532 E73
562 GRP2
541 NC
570 RDY1
726 CONV
633 USEBLK
572 RDY
612 RET
646 AL
670 LINE1
654 NL
667 BEGN
657 SC
722 LOWLA
673 NEWC
705 BLNKS
701 USC
760 HEDRT
1725 PARMSEQ

DISTAPE 66.
LASS
STACK #, FILES>>

13
/0
DISPROG
DISPROG2
BUILDIMAGE
DISPLAY
POTS
;
<RUN>>

CORE USED: 400 2430

<TXT 1
1457 POTS
1402 DISPLAY
1146 NEWIMAGE
1171 ADDC
1304 ADDN
1357 ADDS
6000 J
2000 A
1000 L
4000 S
494 NEW
2017 IMAGE
1047 DISP
1063 VAL
420 M1
424 E1
721 GRP
1101 BLKW
474 BLKG
1111 LINW
535 SKP
522 NORD
1105 BLKIN
520 TP
541 ST7
540 BCD
2013 SKPS
611 NO2
1060 SENSE
556 LINE
1123 TSTW
551 A2
562 A3
602 A4

<TXT 2
1122 HEDR

572 INT
1073 HED
711 BIN
613 INDX
624 X2
634 X3
653 X4
2001 IHED
656 X5
1045 DISP1
714 B2
741 ADT7
737 B3
746 NWL
776 GRP1
755 NHW
1130 TEST2
1002 CHARS
762 CD
1136 ADDB
1007 E73
1034 GRP2
1013 NC
1143 USEBLK
1032 BI1

DISPLAY 68.
<TXT 3
1152 NIR
1153 IM
1154 DUN
1247 IMAX
1210 NL
1217 NCH
1175 ADR
1200 E73
1214 NOC
1205 NEWL
1272 DUN1
1242 NRML
1246 ADD1
1276 E77
1271 E74
1277 NEG
1317 ABS
1310 ADNR
1311 NMBR
1324 NXT
1335 ST
1342 NT
1363 ADSR
1367 ADSN

<TXT 4
1406 DISRET
1413 D1
1417 NEWLINE
1425 NEWCH
1452 BLANK
2221 CODES

<TXT 5
1463 RET
1464 VALSET
1465 NEW
1474 SCNPOTS
1525 TEST
1471 NEWIM
1533 STO
1500 SCRET
1515 SC2
1504 E100
1514 SAMN
2421 POT
1517 E1
LASS
STACK #, FILES>>

14
12
DEFINE
$<RUN>>

CORE USED:  400 1405

<TXT 1
LASS
STACK NO., FILES>

VARBORG
/2
MONITOR
/1
TYPE
LIMTYPEN
T1
INCHAR
;
<RUN>>

CORE USED: 400 2102

<TXT 1

<TXT 2
1735 INCHAR
1624 TYPE
1654 TYPEN

<TXT 3
1710 T1
1651 E277
1653 RTRN
1646 NORMAL
1637 NEWLINE

<TXT 4
1707 RET
1672 ROT

<TXT 5
1721 EM1
1716 LOOP

<TXT 6
1776 RTRN
1745 PULSE
1747 WAIT
1761 IN
177A NOCR

MONITOR
<TXT 1>
1700 TI
1503 NOROOM
1572 SUBST
1602 SB1
1340 NWTRY
1316 FNCH
1631 FINDC
1332 FINDIT
1421 XPD
6000 J
1124 LGL
1361 FRBT
1023 SUB1
635 DISAGN
265 LBUF
335 CAR
6999 GO
20 READ
32 TPIN
404 BLK
33 BN
436 NBLKS
55 AFTP
1213 SETUP
56 WRITE
61 MON
472 BKM ON
65 CODES
346 HED
457 PARS
434 WRT
1435 TMAX
475 DEL
1122 VAL
1216 EI
506 DN
513 DTST
526 DMV
523 DINC

<TXT 2>
534 CLAL
1415 CHNGC
1002 INPTC
541 USE4
543 DISP
1153 FINDL
605 ER
714 AC
705 DISL
712 EM40
661 EMSR
768 TELE
717 NCH
751 DBLK
763 PULSE
765 WAIT
777 IN
1011 NOCP
1375 INCH
1136 CRBT
1206 LNUM
1217 AH
1227 BK
1312 FIND
1363 INPT
1275 CHNG
1237 TYP
1544 PLIN
1320 CMA
1140 OSTS
1152 OTR
1167 TSK

<TXT 3>
1224 PG
1252 TYP T
1271 TCR
1355 FDTR
1536 TYP5
1353 KPCK
1524 RBOT
1414 NCRT
1426 DIFF
1471 XPR
1442 RDIN
1455 EMT
1476 CMV
1545 TYP4

<TXT 4>
1606 SUB2
1615 SUB3
1614 SB2
1634 SB3
1645 JF

<TXT 5>
1711 EM1
1706 LOOP

EDIT 72.
LASS
STACK NO., FILES>

VARBOG
/2
QA
/0
TYPE
LIMTYPEN
T1
INCHAR
;
<RUN>

CORE USED: 400 2077

<TXT 1

<TXT 2
1574 INCHAR
1463 TYPE
1513 TYPEN

<TXT 3
1547 T1
1510 E277
1512 RTRN
1505 NORMAL
1476 NEWLINE

<TXT 4
1546 RET
1531 ROT

<TXT 5
1560 EM1
1555 LOOP

<TXT 6
1635 RTRN
1604 PULSE
1606 WAIT
1620 IN
1627 NOCR
LASS
STACK NO., FILES>

24

VARBORG

2

LASSI

<RUN>>

CORE USED: 400 2775

<TXT 1

<TXT 2

LASS 1

74.
LASS
STACK #, FILES>>

25
/0
VARBORG
/2
LASS2
;
<RUN>>

CORE USED: 400 2200

<TXT 1

<TXT 2

LASS 2
LASS STACK #, FILES>

22
/0
LASS3
TYPE
GENTYPEN
T1
;
<RUN>>

CORE USED: 400 1163

<TXT 1>
1005 TYPE
1035 TYPEN
6000 J
644 PN
442 L0
447 LN
617 SPC
456 RET
6773 RF
464 NXTB
6734 R
6757 SKP
527 NTXT
556 TAG
705 BLINK
711 CR
657 SPAC
760 SYM
704 SPCR
734 READ
754 RR
753 NOTP
773 READF
1004 RFR

<TXT 2
1137 T1
1032 E277
1034 RTRN
1027 NORMAL
1020 NEWLINE

<TXT 3
1150 EM1
1041 RET
1042 VAL
1134 SKIP
1063 BLANK
1102 SPECIFIC
1066 ROT2
1053 ROT1
1070 START
1116 TY1

<TXT 4
1145 LOOP

THIS PROGRAM IS STORED ON BLOCKS 230 AND 237 OF UNIT 0.
NAME/LENGTH/BLOCK NO.

BUILDIMAGE 4 1
CONVRT 1 27
CONVTBL 1 64
DISPLAY 3 5
DISPROG 4 74
DISPROG2 1 31
DISTAPE 4 67
EDIT1 4 40
EDIT2 4 44
EDIT3 4 50
EDIT4 4 54
GENTYPEN 2 10
IMAGE 1 66
INCHAR 2 12
INREC 4 14
LASS3 4 34
LASTCELL 1 20
LIMTYPEN 2 21
POTS 2 23
PROGSTACK 1 25
T1 1 26
TIFAST 1 30
TYPE 2 32
TYPROG 4 60
TYPROG2 1 65
VARBORG 1 73

SYMBOLIC ASSEMBLY PROGRAMS

NAME/LENGTH/BLOCK NO.

BLINK 10 1
DEFINE 10 11
LASS1 10 21
LASS2 10 31
MONITOR 10 51
OA 10 61

BLINK FILES

PARAMS>>

/0

/2
**BUILDIMAGE**

GLOBAL NEWIMAGE ADDC
GLOBAL ADDN ADDS
NEWIMAGE LDA 0
JMP PARAMS (*IMAGE*DONE)
7402
NIR

<10>
,IM
,DUN
LDAV
4000
ADM
IM
STC 17
LDA 17(GET LENGTH)

<20>
ADD IM
COM
STC IMAX
STC NL
STC NCH
JMP NIR
,ADDC LDA (CHAR)
0

<30>
JMP PARAMS
7601
,ADR
LDA; 17
SAE;
,E73 73
JMP NOC
LDA;

<40>
-1
STC NCH
,NEWL LDA;
1
ADM;
,NL
SHD;
1600

<50>
JMP DUN1
,NOC LDA;
1
ADM;
,NCH
SAE;
31
JMP NRML

<60>
LDA
NL
SHD;
1500
JMP DUN1
SET; 15
E73
JMP ADD1

<70>
SET; 15
E77
JMP ADDR1
LDA;
1
STC NCH
JMP NEWL
, NRML SET 15

<100>
17
JMP ADDR1
JMP ADDR
,ADDC LDA;
,IMAX
ADD IM
AZE;
JMP 0

<110>
SET 14
IM
LDA 15
SHD;
7500
JMP DUN1
STH; 14
SHD;
**CONVRT**

GLOBAL CONVTBL CONVRT

CONVRT SET; 17

6400

SET 15

0

NXT LDH; 17

ROR 1

<10>

ADA;

CONVTBL

STC 16

LDH 16

STH 17

LDA

17

SAE

<20>

5

JMP NXT

LDA;

73

STH; 17

LDA;

74

STH; 17

<30>

JMP 15
**CONVTBL**

GLOBAL CONVTBL
VARB
,CONVTBL 3721
2223
2425
2627
3031

<10>
2035
0701
3703
0017
6364
6566
6770
7172

<20>
7314
1037
3737
1552
5354
5556
5760
6162

<30>
3704
1237
3737
1341
4243
4445
4647
5051

<40>
3716
1137
3737
**DISPLAY**

GLOBAL DISPLAY

,*DISPLAY LDA
0
JMP PARAMS
7601
,*DISRET(RETURN LOCATION)
LDA$17

<10>
ADAJ
4000
STC D1+1
,*D1 SET:12
(LOC OF STRING)
SET:13
1340
,*NEWLINE SET:1

<20>
20
LDAJ
-40
ADM
13
,*NEWCH LDH:12
AZEJ
JMP BLANK

<30>
SHD; 7
4000
JMP DISRET
SHD; 7
3000
JMP NEWLINE
ROL 1
ADAJ

<40>
CODES
STC 14
LDAJ
4
ADD 1
STC 1
ADD 13
DSC 14

<50>
DSC: 14
JMP NEWCH
,*BLANK LDAJ
24
ADD 1
STC 1
JMP NEWCH
VARB

<60>
,*CODES 0
0
7500
0
6000
6000
2476
6624

<70>
20
7731
2245
231
1304
1007
7
6000
60

<100>
SHDJ
3600
41
4100
36
3625
2517
3704
404

<110>
CODES
STC 14
LDAJ
4
ADD 1
STC 1
ADD 13
DSC 14

82
<table>
<thead>
<tr>
<th>&lt;120&gt;</th>
<th>&lt;170&gt;</th>
<th>&lt;240&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>4136</td>
<td>4177</td>
<td>3663</td>
</tr>
<tr>
<td>3641</td>
<td>3641</td>
<td>6336</td>
</tr>
<tr>
<td>2101</td>
<td>5177</td>
<td>770</td>
</tr>
<tr>
<td>177</td>
<td>4141</td>
<td>7007</td>
</tr>
<tr>
<td>4526</td>
<td>5077</td>
<td>4543</td>
</tr>
<tr>
<td>3145</td>
<td>4050</td>
<td>6151</td>
</tr>
<tr>
<td>5141</td>
<td>4136</td>
<td>0</td>
</tr>
<tr>
<td>2651</td>
<td>6745</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;130&gt;</th>
<th>&lt;200&gt;</th>
<th>&lt;250&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1070</td>
<td>1077</td>
<td>301</td>
</tr>
<tr>
<td>1337</td>
<td>7710</td>
<td>1707</td>
</tr>
<tr>
<td>5171</td>
<td>7741</td>
<td>2000</td>
</tr>
<tr>
<td>4651</td>
<td>41</td>
<td>3045</td>
</tr>
<tr>
<td>5136</td>
<td>4142</td>
<td>2000</td>
</tr>
<tr>
<td>1611</td>
<td>4076</td>
<td>3045</td>
</tr>
<tr>
<td>4241</td>
<td>477</td>
<td>7777</td>
</tr>
<tr>
<td>7044</td>
<td>2112</td>
<td>7777</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;140&gt;</th>
<th>&lt;210&gt;</th>
<th>&lt;260&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>5126</td>
<td>177</td>
<td></td>
</tr>
<tr>
<td>2651</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>4530</td>
<td>2057</td>
<td></td>
</tr>
<tr>
<td>3446</td>
<td>5720</td>
<td></td>
</tr>
<tr>
<td>1200</td>
<td>1037</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3704</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>4177</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>7741</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;150&gt;</th>
<th>&lt;220&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>1204</td>
<td>4477</td>
</tr>
<tr>
<td>21</td>
<td>3044</td>
</tr>
<tr>
<td>1212</td>
<td>4136</td>
</tr>
<tr>
<td>1212</td>
<td>3743</td>
</tr>
<tr>
<td>2100</td>
<td>4477</td>
</tr>
<tr>
<td>412</td>
<td>3146</td>
</tr>
<tr>
<td>2000</td>
<td>4531</td>
</tr>
<tr>
<td>3045</td>
<td>2245</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;160&gt;</th>
<th>&lt;230&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>7740</td>
</tr>
<tr>
<td>3056</td>
<td>4040</td>
</tr>
<tr>
<td>4437</td>
<td>177</td>
</tr>
<tr>
<td>3744</td>
<td>7701</td>
</tr>
<tr>
<td>5177</td>
<td>374</td>
</tr>
<tr>
<td>2651</td>
<td>7402</td>
</tr>
<tr>
<td>4136</td>
<td>275</td>
</tr>
<tr>
<td>2241</td>
<td>7502</td>
</tr>
</tbody>
</table>
**DISPROG**

GLOBAL POTS DISPLAY
GLOBAL NEWIMAGE ADDC
GLOBAL ADDN ADDS
DEFINE J JMP
DEFINE A ADD
DEFINE L LDA
DEFINE S STC

<10>
SET 1
POINTER
L 1
S POINTER
!NEW J NEWIMAGE
IMAGE
J DISP
L

<20>
3VAL
ADA;
-10
APO;
CLR
ADA;
10
ADA;

<30>
>M1 -1
APO
CLR
ADA;
,E1 1
COM
S GRP
A VAL

<40>
ROL 4
A 2VAL
ADA;
-1777
APO;
CLR
ADA;
1777

<50>
AZE;
CLR
STA
BLKW
SNS:0
S PLKG
L
4VAL

<60>
ROL 3
A 6VAL
A M1
APO
CLR
A E1
S LINW
A 5VAL

<70>
ROL 3
A 7VAL
ADA;
-777
APO;
CLR
ADA;
777

<80>
,M1 -1
APO
CLR
ADA;
,E1 1
COM
S GRP
A VAL

<90>
AZE;
CLR
S SKP
L;
SNS 2
S SENSE
AZE;
J 3
APO

<100>
AZE;
J RCD-2
J ADDS
SKPS
J ADDN
,SKP
SNS 2 (BCD?)
J N02 (NO)

<110>
J NORD
SAE
BLKIN
J 2
J NORD
ROLJ3 (UNIT)
SCR 3
BSE;

<120>
7000
S TP 1
ROLJ4 (GET UNIT)
ADA;
RDC
S TP
A BLKG
S BLKIN

<130>
,S TP RDC
A
,S NORD L;
6777
A SKP
FOR 1
S ST7
A SKP

<140>
AZE;
J RCD-2
J ADDS
SKPS
J ADDN
,SKP
SNS 2 (BCD?)
J N02 (NO)

<150>
,RCD SET;7
,ST7
L;
SNS 2
S SENSE
L;
1
S LINE

<160>
J TSTW
,A2 LDH;7
SAE;
74
J A3
L;
,LINE
A M1

84
S LINW
J BCD
,A3 L
LINE
SAE (LINE FOUND?)
LINW
J+2 (NO)
J A4 (YES)

<200>
A E1
S LINE
,INT SNS 2 (MODE CHNG?)
J NEW (YES)
LDH 7
SHD;
7300 (FIND CAR. RET.)
J A2

<210>
LDH17
J INT
,A4 J HED
LDH 7
S.2
J ADDC
(CHAR)
LDH17

<220>
J+4
,NO2 SNS 3 (INDX?)
J BIN (NO)
,INDX SET 7
ST7
L;
SNS 3
S SENSE

<230>
L;
1
S LINE
J TSTW
,X2 LDH;7
AZE
J X3
L

<240>
LINE
A M1
S LINW
J INDX
,X3 L
LINE
SAE
LINW

<250>
J+2
J X4
A E1
S LINE
LDH 7
ADA;
2
ROR 1
A M1

<260>
ADM
7
J X2
,X4 J HED
J ADDS
IHED
,X5 J ADDC
73

<270>
LDH 7
COM
S 2
J ADDC
XSK;1

<280>
LDH 7
COM
S 1
LDH;7
S.2
J ADDC
LINW
J B2

<290>
LDH 7

<300>
J+5
J ADDC
""
LDH;7
S.2
J ADDN
(LGTH)
J ADDC

<310>
",""
LDH;7
S.2
J ADDN
(BLK)
LDH;7
AZE
J X5

<320>
J DISPL
,BIN LJ
SNS 4
S SENSE
,B2 J TSTW
L
LINW

<330>
MUL;
,GRP
COM
A SKP
STA

<340>
APO
J B3
L;
-1
J R3

<350>
J B3 J HED

<360>
J ADDT
ROR 1
ADA;
7377
S 7

<370>
J NLW SET 1
GRP
<360> SET 2 7 L; 5252 S GRP1 <NHW LDH>2 J TEST2 J CHARS

<370> LDH 2 SCR 3 ,CD ADA; "O" S.2 J ADDC (DIG) LDH 2

<380> BCL; 7770 SRO; 2525 J CD SRO; ,GRP1 J ADDB

<390> XSK;1 J NHW CHAR SET 1 GRP SET 2 J ADDC ,E73 73

<400> SET 2 7 J ADDC 73 AZE APO; J USEBLK

<410> J TEST2 J CHAR S GRP2 <NHW LDH>2 J ADDC (DIG) LDH 2

<420> L; 5252 S GRP2 <NC LDH>2 J TEST2 J DISPI LDH 2 ADA;

<430> -73 AZE APO; J USEBLK A E73 AZE APO; CLR S.2

<440> J ADDC (CHR) J ADDB J NC

<450> J ADDC 73 SRO; J ADDA J ADDC (CHR) J ADDC

<460> J ADDC 73 SRO; J ADDC 74 J ADDC 73

<470> J ADDC 73 SRO; J ADDC 73 J ADDC 73
**DISPROG2**

J 11
USEBLK J ADDC
0
S HEDR
J ADDC
"(
J ADDN
BLKW

<10
J ADDC
")"
J ADDN
BLKIN -1
J ADDC
""
J ADDN
LINW

<20
J ADDC
73
L
BLKIN
AZE;
J .3
APO
J DISP1

<30
HEDR
,TSTW L
LINW
AZE
J
J DISP1
,TEST2 L
2

<40
SAE;
2000
XSK; 0
J
,ADDB SET 11
0
J ADDC
""
**DISTAPE**

GLOBAL INREC REWIND
GLOBAL NEWIMAGE ADDC
GLOBAL ADDN ADDS
GLOBAL DISPLAY POTS
GLOBAL LASTCELL IMAGE
GLOBAL CONVBL
DEFINE J JMP

<10>
J.2 (NO RWD 1ST TIME)
,READ J REWIND
OPR 4 (CLR INPT)
J POTS
POT=1
J.1
LDA;
,E1 1

<20>
STC REC
,MOV CLR
STC NEWIM 1
J INREC
,REC
LASTCELL
7
J ERR

<30>
,NEWIM SRO;
0
J ERR
J HDR
,CHOOSE SNS 3 (BCD,BIN?)
J BCD
LDA;
SNS 3

<40>
STC SENSE
LDA
3POT
ADA;
-10
APO;
CLP
ADA;

<50>
10
AZF
ADD E1
COM
STC GRP
ADD 4POT
POL 5
ADD 6POT

<60>
ADD E1
,SIL STA
LINE
ADA;
,EM1 -1
MUL
GRP
COM

<70>
ROR 1
ADA;
LASTCELL
STC 2
J TEST2
J LOWL
J ADDN
,LINE

<80>
,NEWIM SRO;
0
J ERR
J HDR
,CHOOSE SNS 3 (BCD,BIN?)
J BCD
LDA;
SNS 3

<90>
STC SENSE
LDA
3POT
ADA;
-10
APO;
CLP
ADA;

<100>
,NXTL SET 3
2
SET;4
,GRP
J ADDC
73
LDA;
5252

<110>
STC GRP1
J.2
,NHW LDH;2
J TEST2
J CHARS
LDH 2
SCR 3
,CD ADA;

<120>
"0"
STC 2
J ADDC
(DIGIT)
LDH 2
BCL;
7770
SRO;

<130>
2525
J CD
SRO;
,GRP1
J ADDB
XSK;4
J NHW
,CHARS SET 4

<140>
GRP
J ADDC
,E73 73
,SET 2
3
LDA;
5252
STC GRP2

<150>
J.2
,NC LDH;2
J TEST2
J RDY1
J CONV
ADA;
-73
AZE

<160>
APO;
J USERBLK
ADD E73
AZE;
CLR
STC 2
J ADDC
(CHAR)
<360>
-40-1000
STA
REC
STC.2
J ADDN
(REC)
J ADDC
"",""

<370>
,HEDRT
,FFR CLR
COM
STC NEWIM
J HEDR
J ADDS
PARMSG
J CHOOSE

<400>
,POT
DITTO 7
VARR
,PARMSG 4
"PA"
"R"
"FR"
"R"

<410>
7474
**EDIT1**

GLOBAL T1
GLOBAL NOROOM
GLOBAL SUBST
GLOBAL SB1
GLOBAL NWTRY FNCH
GLOBAL FINDC
GLOBAL FINDIT XPD

<10>
GLOBAL J LGL FRBT SUB1
GLOBAL DISAGN LRUF CAR
DEFINE J JMP
DEFINE GO J
ORG 20
,READ LDA;
RDC 10
STC TPIN

<20>
LDA; 4000
ADD BLK
STC BN
ADD NBLKS
COM
STC 17
,TPIN RDC WRC 10

<30>
,BN
LDA; 1001
ADM
BN
XSK:17
JMP: -7
LDA;

<40>
74
STH
7777
SETJ: 3
6000
LDH: 3
SAE;
74

<50>
JMP: -3
JMP CAR
,AFTP JMP SETOP
,WRITE LDA;
WRC 10
J READ 2
,MON LDA;
J BKMON

<100>

<120>
291
1004
4136
3641
2191
177
4526
3145

<130>
5141
2651
1070
1037
5171
4651
5136
1611

<140>
4241
7044
5126
2651
4530
3446
1200

<150>
100
12
1204
21
1212
1212
2100
412

<160>
2000
3045
2000
3056
4437
3744
5177
2651
<table>
<thead>
<tr>
<th>&lt;170&gt;</th>
<th>&lt;240&gt;</th>
<th>&lt;310&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>4136</td>
<td>275</td>
<td>APO</td>
</tr>
<tr>
<td>2241</td>
<td>7502</td>
<td>J 336</td>
</tr>
<tr>
<td>4177</td>
<td>3663</td>
<td>ADA41</td>
</tr>
<tr>
<td>3641</td>
<td>6336</td>
<td>4000</td>
</tr>
<tr>
<td>5177</td>
<td>770</td>
<td>STA</td>
</tr>
<tr>
<td>4141</td>
<td>7007</td>
<td>.3</td>
</tr>
<tr>
<td>5077</td>
<td>4543</td>
<td>STC WRT</td>
</tr>
<tr>
<td>4050</td>
<td>6151</td>
<td>RDC 10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;200&gt;</th>
<th>&lt;250&gt;</th>
<th>&lt;320&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>4136</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6745</td>
<td>0</td>
<td>SFT12</td>
</tr>
<tr>
<td>1077</td>
<td>301</td>
<td>2000</td>
</tr>
<tr>
<td>7710</td>
<td>1707</td>
<td>LDH12</td>
</tr>
<tr>
<td>7741</td>
<td>2000</td>
<td>SHD12</td>
</tr>
<tr>
<td>41</td>
<td>3045</td>
<td>&quot;AA&quot;</td>
</tr>
<tr>
<td>4142</td>
<td>2000</td>
<td>J WRT 1</td>
</tr>
<tr>
<td>4076</td>
<td>3045</td>
<td>LDA12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;210&gt;</th>
<th>&lt;260&gt;</th>
<th>&lt;330&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>477</td>
<td>7777</td>
<td>&quot;AA&quot;</td>
</tr>
<tr>
<td>2112</td>
<td>7777</td>
<td>STH2</td>
</tr>
<tr>
<td>177</td>
<td>LBUF</td>
<td>LDA12</td>
</tr>
<tr>
<td>101</td>
<td>DITTO 47</td>
<td>J 74</td>
</tr>
<tr>
<td>2057</td>
<td>CAR SET 13</td>
<td>STH12</td>
</tr>
<tr>
<td>5720</td>
<td>0</td>
<td>WRC 10</td>
</tr>
<tr>
<td>1037</td>
<td>LDA12</td>
<td>WRT</td>
</tr>
<tr>
<td>3704</td>
<td>345</td>
<td>LDA12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;220&gt;</th>
<th>&lt;270&gt;</th>
<th>&lt;340&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>4177</td>
<td>J T1</td>
<td>LDA12</td>
</tr>
<tr>
<td>7741</td>
<td>LDA12</td>
<td>ADA12</td>
</tr>
<tr>
<td>4477</td>
<td>353</td>
<td>-4</td>
</tr>
<tr>
<td>3044</td>
<td>J T1</td>
<td>APO12</td>
</tr>
<tr>
<td>4136</td>
<td>J 13</td>
<td>CLR12</td>
</tr>
<tr>
<td>3743</td>
<td>HED&quot;( &quot;</td>
<td>ADA12</td>
</tr>
<tr>
<td>4477</td>
<td>) &quot;</td>
<td>4</td>
</tr>
<tr>
<td>3146</td>
<td>0</td>
<td>STA12</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;230&gt;</th>
<th>&lt;300&gt;</th>
<th>&lt;350&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>5121</td>
<td>0073</td>
<td>NBLKS</td>
</tr>
<tr>
<td>4651</td>
<td>ORG 4000</td>
<td>ROL 10</td>
</tr>
<tr>
<td>7740</td>
<td>J GET</td>
<td>ADA12</td>
</tr>
<tr>
<td>4040</td>
<td>J PARS</td>
<td>1777</td>
</tr>
<tr>
<td>177</td>
<td>J GETCL</td>
<td>COM</td>
</tr>
<tr>
<td>7701</td>
<td>LDA12</td>
<td>STC TMAX</td>
</tr>
<tr>
<td>374</td>
<td>BLK</td>
<td>RDC</td>
</tr>
<tr>
<td>7402</td>
<td>AZE</td>
<td>160</td>
</tr>
</tbody>
</table>
[The image content appears to be a screenshot of a document containing a series of lines of text, possibly computer code or instructions. The text is not clearly legible due to the quality of the image and the style of handwriting. The content includes codes like `J READ`, `PARS ADD 0`, `J LSTOP`, `LDA;`, `BLK`, `J LSTEL 3776 LDA;`, `NRLKS J LSTFL 3776 J LSTCL J PKMON RDC 150 J DEL LDA`, `VAL AZE; ADD E1 COM APO; COM STC 17 STC VAL`, `DN SET 11 2 SET 3 2 LDH 11 DTST SHD; 7400 J DMV`, `SHD; 7300 J DINC LDH; 11 J DTST DINC XSK; 17 J -3 LDH; 11`.]
**EDIT2**

, CLAL CLR
STC CHNGC
STC INPTC
STC SUB1
STC FINDC
, USE4 LDA
4

<10>
, DISP J FINDL
SET 12
2
SET 14
4
LDA
-10
ADD 4

<20>
J FINDL
SET 5
2
SET 6
4
SET 2
12
SET 4

<30>
14
LDA
STC 3
STC VAL
207
SRO
CHNGC
J T1
SET 11
HED

<40>
LDA
3
RCL
4000
SCR 10
ADA
-3
STH

<50>
2000
ADA
"E0 "0"
STH 11
LDH 11
SET 17
-4
NDG LDA

<60>
4
ROL 3
STA
4
RCL 3
7770
ADD E0
STH 11

<70>
XSK 17
J NGR
LDA
M1 -1
ADD 6
COM
ADD 4
COM

<80>
1340
STC AC
SET 11
HED

<90>
HED 4000
J DISL
ADA
LDA
4000
-4000
ADD 5
STC 11
SRO
INPTC

<100>
LDA
J DISL
XSK 17
J -2
LDA
AC
ADA
-14

<110>
HED 4000
J DISL
ADA
LDA
4000
-4000
ADD 5
STC 11
SRO
INPTC

<120>
LDA
J DISL
XSK 12
J -2
LDA
AC
ADA
-14

<130>
SRO
INPTC
ADD EM 40
SET 17
EM 50 -50
SET 16
0
SXL

<140>
J TELE
DIS 16
XSK 17
J -4
SET 11
LBUF
SRO
INPTC

<150>
J DISL
J DISAGN
DISL SET 15
0
SET 1
20
LDA
EM 40 -40

<160>
ADM
, AC
SET 13
-30
NCH LDH 11
SXL
J TELE
SHD

94
<170>
7300
J 15
SHD;
7400
J DBLK
ROLL 1
ADA;
CODES
STC la
LDA;
4
ADD 1
STC 1
ADD AC
DSC 14
DSC 14

<200>
STC 14
LDA;
4
ADD 1
STC 1
ADD AC
DSC 14
DSC 14

<210>
XSK; 13
J NCH
LDH; 11
SAE;
73
J.-3
DBLK LDA
AC

<220>
DSC;
7777
DSC;
7777
J 15
J TELE SET; 17
1241
CLR

<230>
PULSE BSE;
200
WAIT XSK; 17
J.-1
LZE
J IN
ROR; 1
SET; 17

<240>
1340
J WAIT
J PULSE
J IN ROL; 1
SAE;
345
J NOCR

<250>
LDA;
353
J T1
LDA;
73
J 7
DSC;
J NOCR COM
ADA;

<260>
277
J NCH
SCR 1
AZE;
CLR
SRO;
J INPTC
J IPCH (INPT MODE)
SRO;

<270>
SUB1
J SB1 (SBST MODE)
J LGL (CNTRL MODE)
J CRBT
SHD;

<280>
7300
J LNUM
AZE;

<290>
J LNUM
SHD;

<300>
J LNUM
SHD;

<310>
"RR"
J READ
SHD;
"FF"
J FIND
SHD;
"DD"
J DEL

<320>
J SUBST
SHD;
"WW"
J WRITE
SHD;
"MM"
J MON
SHD;

<330>
J SUBST
SHD;
"TT"
J TYP
SHD;
"LL"
J PLIN
SHD;
"TT"
J CMA

<340>
"AA"
J AH
SHD;
"BB"
J BK
SHD;

<350>
ADA;
"-"/"
APO
J CRBT
ADA;
-10
APO;
J CRBT

95
<360>
ADA;
7
STC
ADD VAL
ROL 3
ADD
STA;
>VAL

<370>
J DISAGN
>LOL APO
J
ADA;
-73
APO;
J
ADA;

<400>
73
XSK;
J
CRBT J QSTS
J USE4
>QSTS LDA
0
STC QTR

<410>
LDA;
201
J T1
LDA;
201
J T1
J CAR
>QTR

<420>
>FINDL APO
CLR
AZE;
ADD E1
COM
STC 11
ADD
STC TSK 2
**EDIT3**

LNUM LDA VAL AZE J DISP J USE4 SETOP SET 4 0

<10> J CLAL ,E! 1 AH LDA VAL AZE STC PG LDA; ,PG 1

<20> ADD 4 J DISP ,BE LDA VAL AZE STC PG ADD PG COM

<30> ADD 4 J DISP ,TYP J CAR LDA VAL AZE; ADD M1 APO;

<40> COM STC 11 SET 12 LDH 12 ,TYPT SHD; 7400 J DISAGN

<50> SNS; 1 J DISAGN SHD; 7300 J TCR ROL 1 ADA; -277

<60> COM J T1 LDH; 12 J TYPT ,TCR J CAR XSK; 11 J TCR=2 J CLAL

<70> ,CHNG CLR COM STC INPTC COM STC CHNGC STC VAL J CAR SET; 7

<80> ADD 4 J DISP ,TYPT J CAR LDA VAL AZE; ADD M1 APO;

<90> COM STC 11 SET 12 LDH 12 ,TYPT SHD; 7400 J DISAGN

<100> LBUF LDA; 73 STH; 7 J DEL J KPCK J WPTR ,FRBT J QSTS J CLAL ,INPT CLR COM

<110> J SUBST ,FNCH J FINDIT J CLAL ,CMA CLR COM STC FINDC

<120> J FINDL SET; 6 LBUF J FINDIT J CLAL FINDIT LDA; -4000 ADD 2

<130> STC 12 ADD STC FDTR ,NWTRY SET 11 12 LDH; 12 SHD; 7300

<140> XSK; 4 SHD; 7400 J TYPS SET; 1 LBUF ,KPCK LDH; 1 SHD 6

<150> FDTR (12=BEG, 11=END) SHD; 11 J KPCK J WPTR ,FRBT J QSTS J CLAL ,INPT CLR COM

<160> STC INPTC J CAR SET; 7 LBUF LDA; 73 STH; 7 J USE4
<170>
,IPCH STH 7
J LGL
J RRO T
SHD; 100
J CLAL
LDA;
73

<200>
SHD 7
J,3
STH;7
J DISAGN
SET;1
LBUF
J XPD
,NCRT SRO;

<210>
,CHNGC
J CLAL
XSK14
J INPT 4
,XPD LDA (INSRT STR) 1
COM
ADD 7

<220>
STA;
,DIFF
ROL 1 (NCH)
STC 17
ADD
STC XPR
ADD 3
ADA;

<230>
,TMAX
ROL 1 (-ROOM)
ADD 17
APO;
J NOROOM,
,RDIN LDA
DIFF
ADD 3

<240>
STC 17
LDH 3
STH 17
SET 16
3
SET 3
17
LDA;

<250>
,EMT -4000
ADD 17
STC 17
ADD 16
SAE
2
J CMV (CONT. MOV)
LDA

<260>
1
SAE
7
J.2
,XPR
LDH;1
STH 16
LDH;16

<270>
J.-11
,CMV ADD EMT
STC 16
LDH 16
STH 17
J EMT-1
,NOROOM J CAR
LDA;

<300>
253
J T1
LDA;
163
J T1
LDA;
125
J T1

98
LDA:
147
J T1
LDA:
147
J T1
J CLAL
>RBOT LDA; (BKSPAC)

-4000
ADD 7
SAE;
LBUF
STC 7
LDA;
73
STH 7

J DISAGN
>TYP5 SRO
FINDC
J CLAL
SET 4
5
LDA;
>PLIN J CAR

<TYP4 SET;1
-4
LDA
4
ROL 3
STA
4
BCL;

7770
ROL 1
COM
ADA;
237
J T1
XSK;1
J TYP4 2
**EDIT4**

GLOBAL J LGL FRBT
GLOBAL NOROOM
GLOBAL FINDIT XPD
GLOBAL SUBST
GLOBAL SB1
GLOBAL NWTRY FINDC FNCH
GLOBAL SUB1

<10>
GLOBAL DISAGN LBUF CAR
, SUBST CLR
COM
STC SUB1
STC SUB2
STC SUB3
SET; 7
LBUF-4000

<20>
J DISAGN
,SB1 STH; 7
J LGL
J FRBT
SRO;
, SUB2
J SB2
CLR

<30>
COM
STC SUB2
J DISAGN
,SB2 SRO;
, SUB3
J SB3
SHD
LBUF

<40>
J•2
J DISAGN
CLR
COM
STC SUB3
SET 6 (6=MID BRK CHR)
7
SRO;

<50>
,FINDC
J FNCH
J DISAGN
,SB3 SHD 6
J•2 (READY)
J DISAGN
J CAR
LDA;

<60>
-4000
ADD 7
STC 7
STC 5
,JF J FINDIT
LDH 11
SHD 7
J•4

<70>
SHD;
7300
J FRBT
XSK; 5 (1 MORE OCCURANCE)
SET 16
12
SET 17
11

<100>
LDH; 17
STH 16
SHD;
7400
J•3
LDH; 16
J•-6
SET 1

<110>
6
SET 2
12
J XPD
SET 2
16
J JF
**GENTYPEN**

GLOBAL TYPEN T1 E277 EM1

* TYPEN LDA (VAL, NO. DIGITS)
0
JMP PARAMS
7502
*RET
*VAL

<50>
STC 11
COM
STC BLANK
ADD VAL
SET 12
11
ROR 3
XSK:12

<60>
JMP .-2
STC VAL
JMP ROT2
* TY1 SET 12
0
LDA
VAL
ROL 3

<70>
STA
VAL
BCL:
7770
ROL 1
COM
ADA:
237

<100>
SRO;
*SKIP
JMP T1
JMP 12

<30>
BLANK
JMP T1
JMP TY1
* ROT2 XSK:11
JMP ROT1
* START CLR
COM
STC SKIP

<40>
ADD 11
ADD EM1
STC 11
JMP TY1
XSK:11
JMP .-2
JMP RET
*SPECIFIC COM (USE CONSTANT SIZE FIELD)
**IMAGE**

GLOBAL IMAGE
VARB
*IMAGE 120
DITTO 120
7474
**INCHAR**

GLOBAL INCHAR
,INCHAR XSK;0
CLR
ADD 0
STC RTRN
SET;17
1241

<10>
SXL
JMP -1
,PULSE BSF;
200
,WAIT XSK;17
JMP -1
LZE
JMP IN

<20>
ROR;1
SFT;17
1340
SXL
JMP WAIT
JMP PULSE
,IN ROL;1
SAE;

<30>
345
JMP NOCR
LDA;
73
JMP RTRN
,NOCR COM (NO CAR RET)
ADA;
277

<40>
SCR 1
AZE;
CLR
,RTRN JMP
**INREC**

GLOBAL INREC
GLOBAL REWIND
(CALL IS:
INREC<1$MINL,MAXV$ERR>
N : NO. REC'S TO MOV
MINL: 1'ST LOC TO FIL
MAXV: SET TO MAX LOC

(ERR: ERROR EXIT

ENTER REWIND
REWINDS AND SPACES
),INREC LDA 0
JMP PARAMS
7204

<20>
,RET
,REC
,MINT
,MAXV
,PERR
SET 15
MAXV
ADD REC

<30>
AZE;
JMP RET
APO;
COM
STC 12
LDA
MINL
SAE;

<40>
2000
JMP .4
CLR
COM
JMP .3
ADA;
-4000
STC MINL

<50>
ADD REC
APO
JMP GOBKWD
GOFWD JMP FWD
OPR 3 (READ RESET)
JMP AREC
JMP APAR
XSK; 12

<60>
JMP GOFWD 1
JMP STOP
SET 11
MINL
CLR
STC RPAR
CNCH LDH; 11
BCO;

<70>
,RPAR
STC RPAR
LDA 15
SAE 11
JMP CNCH
LDA
RPAR

<100>
SAE
LPAR
JMP PERR
JMP RET
GOBKWD JMP BKWD
SXL 4
JMP .-1
OPR 4

<110>
JMP AREC
SXL 4
JMP .-1
OPR 4
JMP AREC
XSK; 12
JMP .-5
JMP STOP
<120>
SET; 12
-1
JMP GOFWD
"FWD SET 16
0
LDA;
20
JMP BTH

<130>
,RKWD SET 16
0
LDA;
40
,BTH ATR
SET; 17 (5 MS. DELAY)
7543
XSK; 17

<140>
JMP -1
JMP 16
,STOP SET 16
0
SET; 17 (4.4 MS. DELAY)
7564
XSK; 17
JMP -1

<150>
CLR
ATR
SET; 17 (5 MS. DELAY)
7543
XSK; 17
JMP -1
JMP 16
,APAR LDA(ACCEPT A LONG. PARITY)

<160>
0
STC APRET
SXL 4
JMP APWT
,LPF OPR 4 (FOUND)
STA;
,LPAR
,APRET(RETURN)

<170>
,APWT LD;
-100
SNS; 2
ROL 2
STC 17
SXL; 4
JMP LPF
XSK; 17

<200>
JMP -3
JMP APRET
,ATRC LDA
0 (READ A RECORD)
STC ARET
SET 11
MINL
,ARST SNS; 2 (TEST DENSITY)

<210>
JMP HID-2
SXL 4
JMP -1
,LOD OPR 4
XSK 11
STH; 11
SXL; 4
JMP LOD

<220>
SXL; 4
JMP LOD
SXL; 4
JMP LOD
SXL; 4
JMP LOD
SXL; 4
JMP LOD

<230>
SXL; 4
JMP LOD
SXL; 4
JMP LOD
SXL; 4
JMP LOD
SXL; 4
JMP LOD

105
**LASS3**

GLOBAL TYPE TYPEN (
COMMENT QUARTER 1
OF THIS PROG IS
STORED ON BLOCK 230,
QUARTER 2 ON BLK
237; IT IS A THIRD
PASS OF LASS. )

<10>
DEFINE J JMP
LDA 1
STC PN
ADD 2
STC LØ
ADD 3
STC LN

<20>
RDC 2237
J TYPE 73
J TYPE "C"
J TYPE "O"

<30>
J TYPE "R"
J TYPE "E"
J TYPE "U"

<40>
J TYPE "S"
J TYPE "E"
J TYPE "D"
J TYPE ":"

<50>
J TYPE
J TYPEN LØ
J TYPE
J TYPEN

<60>
J TYPE
J TYPE LN
J TYPE
J TYPEN

<70>
J TYPE
J TYPE
J TYPE
J TYPE
J TYPE SPC

<80>
J TYPE
J TYPE
J TYPE
J TYPE
J TYPE

<90>
J TYPE
J TYPE
J TYPE
J TYPE
J TYPE

<100>
J TYPE
J TYPE
J TYPE
J TYPE
J TYPE

<110>
J TYPE
J TYPE
J TYPE
J TYPE
J TYPE

<120>
SHDJ
J NTXT
SHDJ
J NTXT
SHDJ

<130>
J SPJC
J SPJC
J SPJC
J SPJC
J NTXT

<140>
J NTXT J TYPE
J TYPE
J TYPE
J TYPE
J TYPE

<150>
"X"
J TYPE
J TYPE
J TYPE
J TYPE

<160>
LDA;
1
ADM
-5
J NTXT
J NTXT

107
<170>
COM
STC 17
SET 3
17
SET 1
SYM
R
STH 1

<200>
XSK 17
J -3
RF
ADA 1
3000-4000
STC 2
LDA 2
STC 2

<210>
J TYPE
(TAG DEF)
4
J TYPE
SET 1
SYM
LDH 1

<220>
STC 2
J TYPE
(CHAR)
XSK 3
J -5
J TYPE
73
J NXTB

<230>
,SPC J SPAC
RDC 220
SET 1
RGC 1
SET 2
6221
SET 3

<240>
WCG 1
LDA
0
SCR 10
STA
4
COM
STC 0

<250>
ADD LN
SCR 10
ADD 0
ROL 6
ADA 1
,PN
ROL 3
ADM

<260>
4
SET 5
RDC 1
SET 6
1230
SET 7
J RET
J 1

<270>
,SPAC LDA
0
STC SPCR
J TYPE
-26 (LINE FEED)
J TYPE
J TYPE
J NXTB

<280>

<290>

<300>
-26
J TYPE
-26
J TYPE
-26
J TYPE
-26
J TYPE
-26
J TYPE

<310>
-26
J TYPE
-26
J TYPE
,SPCR
,BLINK CLR
COM

<320>
STC CR 1
J NXTB
,CR SRO 1
0
J 2
J NXTB
CLR
STC CR 1

<330>
RF
COM
STC 17
RF
RF
R
XSK 17
J -3

<340>
RF
AZE 1
J NXTB
RF
J -4
,READ LDA
DEFINE R J READ
0

<350>
STC RR
ADD 5
SAE 1
6777
J NOTP
RDC
5201
LDA 1

108
<360>
1
ADM
.-4
SET;5
6377
.NOTP LDH;5
.RR
R(SKP 2 BYTES)

<370>
R(SKP 1 BYTE)
DEFINE SKP J.
J NXTB
.SYM
DITTO 12
.READF LDA
DEFINE RF J READF
0

<400>
STC RFR
R
ROL 6
STC 2
R
ADM
2
.RFR

<410>
**LASTCELL**

GLOBAL LASTCELL
VARB
*LASTCELL
**LIMTYPEN**

GLOBAL T1 TYPEN
    TYPEN LDA; (VAL)
    JMP
    ADD 0
    STC 14 -
    XSK: 0
    ADD 0

<10>
    STC RET
    JMP ROT
    SHD;
    3700
    JMP .-3
    JMP T1
    JMP ROT
    JMP .-2

<20>
    ROT LDA 14
    ROL 3
    STA 14
    RCL;
    7770
    ROL 1
    COM
    ADA;

<30>
    237
    SRO;
    3567
    JMP 0
    JMP T1
    RET JMP
**POTS**

GLOBAL POTS
• POTS LDA
0
JMP PARAMS ($VALS-1$ HAVCHNGD)
7402
• RET
• VALSET

<10>
• NEW
JMP SCANPOTS
JMP TEST
JMP RET
• NEWIM JMP SCANPOTS
JMP STO
JMP NEW
• SCANPOTS LDA

<20>
0
JMP PARAMS
7601
• SCRET
LDA17
STC SC2
LDA1
• E100 SAM 0

<30>
STC SAMN
SET$11$
POT-1
SET$12$
7767
SET 13
VALSET
• SAMN

<40>
• SC2 (JMP TEST OR STO)
LDA1
• E1 1
ADM
SAMN
XSK12
JMP SAMN
JMP SCRET

<50>
• TEST COM
ADA11
SCR 1
AZF
JMP NEWIM
JMP 0
• STO STA$11$
ADA1

<60>
200
SCR 2
STA$13$
JMP 0
VARB
• POT 0
DITTO 7
**PROGSTACK**

```plaintext
<10>
TYPE$: (MSG, FMT, FILE)$
BLINK$: FILE$
DISTAPE$: NIL$
DISPLAY$: NIL$
DEFINE$: 1 STRING$
MNTR$: $
EDIT$: TEXT$

<20>

QA$
LASS$: STACK #, FILES$
$$
```

113
GLOBAL T1 EMI
T1 SET 15
Ø
SNS; 0
JMP; -1
SET; 16
-13

<10>
LOOP STA;
Ø
BCL;
EM1; -1
ATR
SET; 17
7355
XSK; 17

<20>
JMP; -1
LDA
LOOP +1
SCR 1
XSK; 16
JMP LOOP
JMP 15
GLOBAL T1 EM1
,T1 SET 15
Ø
SNS;Ø
JMP.-1
SET;16
-13

<10>
,LOOP STA1;
Ø
BCL1;
,EM1 -1
SNS;2
ROL 1
ATR
LDA1;

<20>
$356
SNS;2
ADD FAST
STC 17
XSK;17
JMP.-1
LDA
LOOP+1

<30>
SCR 1
XSK;16
JMP LOOP
JMP 15
,FAST 364
**TYPE**

GLOBAL TYPE T1 E277
  TYPE LDA:
  1776
  ADD 0
  STC 17
  XSKJ 0
  ADD 0

<10>
  STC RTRN
  LDA; 17
  SAE;
  73
  JMP NORMAL
  NEWLINE LDA;
  345
  JMP T1

<20>
  LDA;
  353
  JMP T1
  JMP RTRN
  NORMAL COM
  ROL 1
  ADA;
  E277 277

<30>
  JMP T1
  RTRN JMP
**TYPROG**

GLOBAL TYPE TYPEN
DEFINE X XSK
DEFINE J JMP
DEFINE A ADD
DEFINE L LDA
DEFINE S STC

(PARAMS ARE:
<10>
    MSG, FMT, BLK, NBS
    REPEATED.
    FMT:
    1=TXT
    2=UNT 1 OCTL
    3=UNT 0 OCTL
    SNS 0 = PAUSE
    SNS 1 = RTRN
<20>
    SNS 2 = ANALG TAPE
)SET 4
POINTER
L 4
STA
POINTER
S 4
J BL

<30>
14
"NWTXT L;4
APO;
J BELLs
SET 1
A
COM
A 4

<40>
S 4
S OCTADE
L;4
L;4
S FMT
L;4
L;4
BSE;

<50>
7000
S 1READ
L;4
L;4
AZE;
J BELLs
COM
S NC

<60>
L;
RDC 10
S READ
L;
,FMT
SHD;
100
J TXT

<70>
SHD;
200
J OCTAL
SHD;
300
J UNT0
J PUT
J BFMT

<80>
<90>
<100>
100
J RETURN
,"BFMT A
J LSTOP
J STRING
.2
J LSTCL
5
"IL"

<110>
"LG"
"L"
"FM"
"T"

<120>
LDH 7
AZE;
ADD NC
APO
COM
ADD NC
APO;
CLR

<130>
COM
A NC
S NC
LDH;7
SAE;
"A"
J OTXT
SET;5

<140>
-67 (55 LINES)
J NAME
J BL
1
SET;3
-7
J LINE
X;3

<150>
J -2
SET;6
-4
,"NWP J LNUM
SET;3
-10
J LINE
X;3

<160>
J -2
X;6
J NWP
J FINP
SET;6
-5
SET;5
-67
**TYPROG2**

J TYPE 0
SNS 1
J STEP
J RESTORE
DITTO 2
J RETURN

<10>
BY3 J STEP
BY2 J STEP
BY1 J FINP
SET: 5
-67
X: 3
J NXB
J NWTXT

<20>
**VARBORG**

VARB

ORG 2000
STANFORD BLINK COMPILER -- VERSION OF 20 MAY 65

COMMENT THIS IS PASS 1 OF THE ASSEMBLERS

GLOBAL NEXIT, GSWS
GLOBAL TXTN$,
GLOBAL CON, MODE, BLK, BASE, CHAR, CONMAX,
T, SEARCH, VALGLR, GLBSW, BLINKSW, LINE, SAVE, BADBLK$
GLOBAL PSEUD, BUFFER$
GLOBAL SRCHEARS()
ARRAY BUFFER (10)$
ARRAY VALGLR (28)$
ARRAY BLK(0..3) = ("6200", "6200", 0, "7200")$
ARRAY CHAR (10)$

COMMENT
PARAMETERS ARE . . . STK NO., TEXT$*$

COMMENT CORE USE PLAN IS .
SYMBOL TABLE ARRAY ETC THRU QTR 5.
GTR 6.. FIRST HALF OUTPUT
QTR 7. .. ALTERNATELY TEXT INPUT AND ORIGINAL QTR 6$

COMMENT EXTERNAL MEMORY PLAN..,
BLOCK CONTENTS
0 ORIGINAL QUARTER 6
1-17 TEMP CODE STORAGE
TEMPORARILY EXT MEM WILL BE SIMULATED BY
TAPE UNIT 0 .. HLKS= (200)+EXT BLK NO.$

11... PROCEDURE TAPE ( INST, INC, N )$
11... BEGIN
13... M(9) = INST$
15... M(10) = INCR(INC, BLK(N))$
19... M(11) = "6000"
21... DO "6011"
27... END TAPE()$
27...$
30... PROCEDURE WRITE(INFO)$
37... BEGIN
41... IF I6 EQL "7377"
42... BEGIN I6 = "6777"
46... TAPE("704", 1, 1)$
52... END$
53... M(161) = INFO$
55... END WRITE()$
55...
58... PROCEDURE WRITEF(INFO)$
55... BEGIN WRITE(SCLR(I6, INFO))$ WRITE(INFO)$
77... END WRITEF()$
77...
100... GLOBAL NEGL$
100... PROCEDURE SEARCH($$FOUND)$
107... BEGIN
107... CCN = M(151)+M(15)$
113... UNTIL M(121) EQL 0$
BEGIN
  REI = CHAR(I)$
  I2N = I2 + (MH(I2) - '0')/2$
  IF M(REI) NEQ M(I2) $ GO NEQL$
  UNTIL MH(REI) EQL O$
  IF M(REI) NEQ MH(I2)$ GO NEQL$
  GO FOUND$
  NEQL, I2 = I2N$
  CON = CON + M(I5)$
  END$
  END SEARCH(I)$

PROCEDURE LOOKUP($$CONTROL, RESERVED, TAG$$)
BEGIN
  COMMENT CHAR IS SOUGHT FIRST AMONG LOCALLY VALID GLOBALS.
  (IF FOUND THERE, SEE 'FOUND')
  IF ABSENT AND GLBSW ON, SYMBOL IS ADDED TO GLOBAL TABLE.
  OTHERWISE SOUGHT IN LOCALS. IN LATTER CASE, ORPHANS ALWAYS
  HOUSED. IN ANY CASE SYMBOL FINALLY 'FOUND'..
  THE VALUE OF A SYMBOL (CON) MEANS . . .
  PSEUDO OP . . SWITCH VALUE
  RESERVED WORD . . 6-BIT POINTER
  IDENTIFIER (TAG) . . FULL-WORD POINTER
$}

ARRAY T('650') = ( 'GLOBAL', 'UPPER', 'DEFINE', 'ORG', '5DITTO', 'VARB', 'CONT', 'HLT', 'CLR', 'MSC', 'ATR', 'RTA', 'NOP = ', 'COM', 'RCL', 'RCR', 'SCR', 'ROR', 'SCR', 'RCL', 'LDA', 'STA', 'ACA', 'ADM', 'LAM', 'MUL', 'SAE', 'SRO', 'BCL',

124.
'3BCO',
'3BSE',
'3DSC',
'3LCH',
'3STH',
'3SHD',
'3SET',
'3SAM',
'3DIS',
'3XSK',
'3OPR',
'3SNS',
'3AZE',
'3APD',
'3LZ',
'3IBZ',
'3SXI',
'3KST',
'3RDC',
'3RCG',
'3RDE',
'3MTH',
'3WRC',
'3WCG',
'3WRK',
'3CK',
'4SAVE',
'7RESTORE',
'6PARAMS',
'6REPEAT',
'4STEP',
'7OVERLAY',
'7POINTER',
'3PUT',
'3GET',
'5GETCL',
'5LSTOP',
'5LSTCL',
'6STRING',
'5LSTEL',
'6RETURN',
0)

ARRAY SRCHPARS(7)=(0,1,256,-1,0,1,0)

COMMENT FIRST SEARCH PSEUDO-OPS...
151=SRCHPARS(1)
121=T(0)
SEARCH($$GOC)

COMMENT NEXT SEARCH RESERVED WORDS...
151=SRCHPARS(1)
SEARCH($$RESERVED)

COMMENT NEXT SEARCH GLOBALS...
151=SRCHPARS(3)
IF GLBSW$ GO HOUSE$
SEARCH($$STSTGLB)$
GO LOC$L$

TSTGLB..
COMMENT SEE IF AN ACTIVIZED GLOBAL $
III=VALGLB(0)$
UNTIL M(III) EQL 0 $ IF M(11) EQL CON $ GO GOT$ 
COMMENT NO SUCH LUCK $
GO NEQL$

COMMENT FINALLY LOCALS..$
II=SRCHPARS(5)$
SEARCH($$GOT)$
M(I)I)=CON$
GO ESYM$

HOUSE..
SEARCH($$GOT)$
ESYM$
WRITE(2)$
II=CHAR(1)$
WRITE(MH(II)-'0')$
MH(I2)=MH(II)$
REPEAT (MH(II) - '0') $ ( MH(I2)=MH(III)$ WRITE(MH(II)) )$

M(I2I)=M(I2I)=M(I2I)=0$
GOT.. GO TAG$
GOC.. IF CON EQL 7$ GO NEXTEXT$

WRITE(8)$
PSEUD=1$ GO CONTROL$

END LOOKUP()$

PROCEDURE RETWR(STR(1)$
BEGIN TAPE(''700'',0,0)$
LIST BBL(STR(),TXTN,LINE)$
M(''21'')=BSAVE$
PUT($$BBL)$
OVERLAY 0$
END RETWR()$

STRING ILTXT(5)=(''ILLGL TXT''),
BDCHR(6)=('BAD CHAR AT'')$
GLOBAL ILTXT,BDCHR,PRSOF$

PROCEDURE SCAN($$CONTROL,RESERVED,TAG,CONST)$
BEGIN COMMENT 17 SCANS TEXT IN QUARTER 7$
FUNCTION BTWN(X,Y)=(X LEQ MH(17)) AND (MH(17) LEQ Y)$
SUBROUTINE STEPS$
BEGIN
IF I7 EQL 0$
BEGIN
COMMENT END OF BLOCKS$
TAPE(''710'',1,2)$
I7 = ''7377''$
END$
I7 = I7 + ''4000''$
RETURN END STEPS$

ARRAY CONS(10)=(''73,73,74,74'','
',',',',',',',',',',',',',',',',',',',')$

TOP$
FOR CON = (1,1,10)$
IF MH(I7) EQL CONS(CON)$
BEGIN ENTER STEPS IF CON GTR 7$ GO TOP$
GO CONTROL$ END$
END

BEGIN CON = 0$
NBCD.. ENTER STEPS$
IF MH(I7) EQL 2$
GO EC$
CON = 64CON + MH(I7)$
GO NBCD$
EC.. ENTER STEPS GS..$ GO CON$)
END$

BEGIN
CON = 0$
UNTIL NOT BTWN('0', '7')$
CON = 8CON + ABM(MH(I7) - '0')$ ENTER STEPS$
END$

BEGIN
MH(II) = MH(I7) + 2(I1 - I2) + '0'$ MH(II) = 0$
LOOKUP($$GOC,RESERVED,TAG)$
END$
OTHERWISE $ RETWR(BDCHR())$ ENTER STEPS$
END$
GO TOP$
END SCAN($
TAPE(""704"",0,0)$
COMMENT QTR & NOW SAVED$
WRITE(M(""21""))$ I11=BUFFER(G)$
BSAVE = M(""21"")$
M(""21"")=I11$
I3=M(POINTER)$
POINTER = I3$
DMY = M(I3I)$
COMMENT SKIP CONTROL WORD IN BUFFER$
WRITE(11)$
WRITE (M(I3I))$
I3=I3+256$
GO CLVGS$

TAPE(""700"",0,3)$
TXTN=TXTN+1$

1000$ DO(223)COMMENT SKIP CONTROL WORD IN BUFFER$
1001$ BLK(2)=BSET(""7000"",M(I3I))$
1006$ IF M(I3) EQL 0$ GO PASS2 COMMENT NO MORE TEXTS$
1011$ LINE=0$
1014$ WRITE(12)$ I7 = ""3401""$
1020$ DO(""223")COMMENT SKIP CONTROL WORD IN BUFFER$
1022$ DO(""223")COMMENT IGNORE LENGTH OF TEXTS$
1026$ TAPE(""710"",0,2)$
1026$ EITHER IF MH(""7400") EQL 'A' COMMENT STANDARD ALPHA-TEXT$
1037$ BEGIN BLINKSW= 0 $ END$
1037$ OR IF MH(""7400") EQL 'B' COMMENT A BLINK TEXT$
1053$ BEGIN GLBSW=BLINKSW=1$ WRITE(11) GO NEXTSYM ENDS$
1056$ OTHERWISE $ RETWR(IILTXT(1))$
1062$ NEXTLINE..$
1062$ GLBSW=0$
1064$ LINE=LINE+1$
1071$ IF BLINKSW$
1073$ BEGIN
1073$ COMMENT PROCESS NUMERIC PART OF BLINK TEXT$
1073$..$
1101$ SUBROUTINE BW$
1101$ BEGIN
1102$ IF I7 EQL ""7777""$
1102$ BEGIN TAPE(""710"",1,2)$
1110$ I7=""7377""$
1112$ END$
1112$ WRITE(MH(I7I))$
1116$ RETURN END BW$
1117$..$
1117$ PROCEDURE FW(DMY)$
1126$ BEGIN
1126$ GLOBAL BW$
1126$ ENTER BW$
1127$ PT1=MH(I7)$
1132$ ENTER BW$
1133$ FW()=64PTI+MH(I7)$
1142$ END FW()$
1142$..$
1143$ I7=I7-""4000""$
1147$ REPEAT FW(0) $ END$
1155$..$
FW(0) $  COMMENT 2+3FW(0) OVERFLOWS LINC WORD $  
VRBWS.. IF FW(0) EQL O$ GO NEXTEXTS  
FW(0)$  
GO VRBWS$  
ENDS  
NEXTSYM..  
PEUSD=0$  
SCAN($$CONTROL,RESERVED,TAG,CONST)$  
RESERVED..  
WRITE(9)$ WRITE(CON)$ GO NEXTSYM$  
TAG.. WD=BTSET(''4000'',$ CON)$  
IF GLBSW$ (M(14) =CON$ M(141) =0)$$  
WTAG.. WRITF(WD)$  
GO NEXTSYM$  
CONST.. WRITE(10)$ WD=CON$ GO WTAG$  
CONTROL..  
WRITE(10)$  
EITHER IF PSEUD$ CON = CON +4$  
OTHERWISE$ CON=MIN(CON,4)$  
SWITCH CON, (NEXTLINE,EOT,EOT,EOT,NEXTSYM,GLB)$  
COMMENT CONTROL OF MEANING ONLY TO SECOND PASS $  
GO NEXTSYM $  
EOT.. COMMENT END OF A TEXT $  
COMMENT ERASE LOCAL SYMBOLS $  
I2I=T(0)$ CHAR(1) =0$  
EOT1.. SEARCH($$EOT1)$ COMMENT CONTROLS$  
EOT2.. SEARCH($$EOT2)$ COMMENT RESERVED WORDS $  
EOT3.. SEARCH($$EOT3)$ COMMENT GLOBALS $  
M(12I)=M(12I)=0$  
SRCHPARS(5) = SRCHPARS(7)$  
CLVGS.. I4I = VALGLB(0)$  
M(141) = 0$  
GO NEXTEXT$  
GLB.. GLBSW=1$ GO NEXTSYM$  
PASS2..  
I1 = ''3400'' - 1$ REPEAT 256$ M(111) = 0$  
WRITE(4)$  
TAPE(''704'',''1,1)$  
OVERLAY -''25''$  
COMMENT ON TO SECOND PASS $  
FINISH$
**STANFORD BLINK COMPILER -- VERSION OF 20 MAY 65**

---

**-SPACE**

**COMMENT** THIS PROGRAM IS PASS 2-4 OF THE ASSEMBLER.

**PASS 2**.. DEFINE TAGS AND DETECT ERRORS,

**PASS 3**.. ASSEMBLE CODE-SPACE,

**PASS 4**.. ASSEMBLE VARB-SPACE. AND EXEUNT.

**INDEX REGISTER ALLOCATION..**

1.. FREE
2.. FREE
3.. POINTS TO LOCATION COUNTER
4.. BLINK TEXTS
5.. SCAN PASS 1
6.. BLINK TEXTS
7.. OUTPUT

**FIRST PASS CODES, WITH COUNT OF FOLLOWING DESCRIPTION BYTES..**

1 (0) END OF LINE  9 (1) RESERVED WORD
2 (0) TAG NAME    10(2) CONSTANT   32+(1) TAG POINTER
3 (0) END OF TEXT 11(2) PROG. NO.
4 (0) END OF OUTPUT 12(2) IN TEXT
5 (0)
6 (0)
7 (0)
8 (1) PSEUDO-OP

**PSEUDO-OPS . . .**

'GLOBAL' . . . TAGS BECOME COMMON WITH OTHER TEXTS.
'UPPER' . . . ',' AND '.' HAVE 2000 SUBTRACTED FOR
UPPER CORE.
'DEFLNE' . . . A TAG IS GIVEN A VALUE.
'ORG' . . . LOCATION COUNTER IS SET
'DITTO' . . . PREVIOUS LINE IS REPEATED
'VARB' . . . FOLLOWING CODE IS TACKED ON AT END

**COMMENT ERROR INFORMATION IS . .**

**DUPLICATE TAG . . . BLOCK NUMBER AND LINE NUMBER OF OCCUR.**

**ILLEGAL OPERATION . . . "" "" ""**

**UNDEFINED TAG . . . "$",

**COMMENT** PROG IS ASSEMBLED INTO STACK LOCATION
22.. A ROUTINE IN BLOCK 230 MOVES IT
TO THE REQUESTED LOCATION$.

**COMMENT STORAGE ALLOCATION..**

QTR 4.. ARRAYS, QTR 5.. READ 1ST PASS,
QTR 6.. SYMBL DEFS
QTR 7.. DEFBIT FLAGS/OUTPUT

$ COMMENT SAVE-BUFFER IS KEPT IN AN ARRAY TO FREE QTR 6$

ARRAY SAVBUF(10) $
GLOBAL UPPER $ COMMENT FACILITATES USE OF UPPER CORE $  
GLOBAL PROGNUM$  
GLOBAL SCNO, SKPEQL, ONAM, LINE, CON, BADOP, BADOP1, UNDEF, LSTWRD $  
GLOBAL BLK, PASS, VARB, VRBSW, VABS, ABSL, LOCN, ERSW, 
NOWRITE, WORD, UNDEF, EOT, E00, BLINK, CON, NOPS, LOCNO $  
GLOBAL NAM, BLOCK$  
GLOBAL UNDTAG, UNTGMSG$  
ARRAY BLK(5..7) = (''5200'', ''6200'', ''7217'')$  
PROCEDURE TAPE ( INST, INC, QTR ) $  
11... BEGIN  M(9) = INST $  M(10) = INCR( INC , BLK(QTR)) $ 20... M(11) = ''6000'' $  DO ''6011'' $ 27... END TAPE() $ 27...  
30... COMMENT INITIALIZE INPUT TAPE $  
30...  
32... PROCEDURE READ(DMY) $  
41... BEGIN  
41... IF I5 EQL ''6777'' $  
43... BEGIN  TAPE(''700'', 1, 5 ) $  I5 = ' 0377' $  END $  
53... READ() = MH(I51) $ 54... END READ() $ 54... 
54... 
55... PROCEDURE READF(DMY) $  
64... BEGIN  
73... READF() = 64.READ(0)+READ(0) $  
74... END READF() $ 74... 
74... 
75... PROCEDURE IFN2($$$L)$(IF PASS NEQ 2 $ GO L )$ 112... 
116... PROCEDURE WRITE(INFO) $  
125... BEGIN  
125... INCR(1, M(I3))$  
127... IF PASS - VRBSW NEQ 3 $ RETURN $  
140... IF I7 EQL ''3777''$  
143... BEGIN  TAPE(''704'', 1, 7) $  I7 = '3377' $  END $  
153... LSTWRD=M(I71)=INFO$  
160... END WRITE() $ 160... 
160... 
161... PROCEDURE SCAN( $$ TAG , CONST , RESRVD , PSEUDO ) $ 170... BEGIN  
170... ARRAY VALUE ( 63 ) = ( ''  
170... 0, 11, 0, 14, 15, 16, 17, 240,  
170... 300, 340, 2000, 4000, 6000, 1000, 1040,  
170... 1100,11-0, 1200, 1240, 1440,  
170... 1500, 15-0, 1640, 1600, 1740, 1300,  
170... 1340, 1400, 40, 100, 140, 200, 500, 440,  
170... 450, 451, 452, 453, 400, 415, 700, 701,  
170... 702, 703,704, 705, 706, 707,  
170... 20, 46, 74, 123, 146,  
170... 343, 155, 164, 214, 235,  
170... 226, 241, 244, 263, 336''  
170... )$  
170... SGN=1$ UNDEF=0$  
173...
TOP.. IF READ(O) LSS 32 $ SWITCH MH(I5) ,
(EOL, SKP, SCMN, EDO, DOT, COMMA, PSEUM, PSEU1,
RSRVD, CONSTNT, PROGN, INTXT, BLINK
$)

COMMENT A TAG $
CON= SC LR(I1, ROTR (5 , MH(I1))) +READ(O) $
I1 = CON + "3000" $ 12=I1 + 256 $
IF I2 EQL UNTAG$ GO UNTMSG$
WORD = WORD + SGN.M(I1)$
IFN2( $ GOT )$
M(I2)= MAX ( M(I2) , 1 )$
GOT.. GO TAG$
DOT.. UNDEF = 1 - ABSL$
WORD=WORD+SGN(M(13)-UPPER) $
GOR.. GO RSRVD $
CON.. CON=7$
GDP.. GO PSEUDO $
PSEUM.. SGN = -SGN $ GO TOP $
RSRVD.. WORD=WORD+ SGN.VALUE(READ(O)) $ GO GOR $
CONST.. CON= SGN.READF(O) $ WORD=WORD+ CON $
GO CONST$
SKP.. REPEAT(READ(O))$ READ(O)$ GO TOP$ 
PROGN.. PROGNUM=READF(O)$
GO TOP$
INTXT.. BLOCK = BLOCK + 1$
LINE = 1$
GO TO TOP$
PSEUI.. CON = READ(O)$ GO GOP$
EOL.. LINE= LINE + 1 $
END SCAN($) $
STRING DUPL(7)=('DUPL. TAG AT'),
UNDF(8)=('UNDEF. TAG AT'),
ILLGL(6)=('ILLGL OP AT')$
GLOBAL GOODAS, DUPL, UNDF, ILLGL, STK $
PROCEDURE RETWR(STR())$
BEGIN LIST L(STR(), BLOCK, LINE)$
TAPE("700", 0, 6)$
GLOBAL BSAVE $
M("21") = BSAVE $
PUT($$L)$
OVERLAY O$
END RETWR($) $

PROCEDURE DEFEXPR(DMY)$
BEGIN WORD= 0 $
NXT.. SCAN ($$ TAG, NXT, RSRVD, BADOP) $
DEFEXPR() = WORD $ RETURN $
TAG.. IFN2( $$ NXT )$
IF M(I2) NEQ 2 $ GO BADOP $ GO NXT$
RSRVD.. IF UNDEF $ GO BADOP$ GO NXT$
END DEFEXPR()$

SUBROUTINE GETAG$
BEGIN
SCAN ( $$ TAG, BADOP, BADOP, BADOP )$

132.
GO BADOP $
$TAG.. IFN2( $$ RET ) $ IF M(12) NEQ 1$ RETWR(DUPL())$ M(12) = 3 - ABSL $ RET.. 14=11$ RETURN $ END GETAG $

ARRAY BLNKS(20) $

COMMENT GET READY FOR PASS 2 $
PASS=2$

COMMENT DEFINE FLAGS.. 1.. UNDEFINED 2.. DEFINED 3.. ADD VARR $

GLOBAL ARRAY LOCN(0..1) =(''400'') $ LOCNO = 256 $

COMMENT LOCN(0) INDEXES CODE SPACE, LOCN(1) VARB SPACE. VABS DENOTES ENCOUNTERING OF 'ORG' IN VARB SPACE $

SCM2..BSAVE=READF(0)$ 11I=SAVBUF(1)$ M(''21'')=I1S BLOCK = 0$

SCMO.. VRBSW = 0 $ 13I = LOCN(0) $ ABSL = 1 $ UPPER = 0$

SCO.. NOPS = WORD = 0 $

SC.. SCAN ( $$ SC , SC , SC , PSEUDO ) $ WRITE (WORD) $ GO SCO $

SC. UNTGMSG.. RETWR(UNDF())$ PSEUDO.. SWITCH CON, (GLB,UPR,DEF,ORG,DIT,VRB,COMMA)$

GLB.. SCAN($$ GLB,BADOP,BADOP,BADOP)$ GO SCO$

UPR.. UPPER=''2000''$

BAS.. COMMENT EXPECT CARRIAGE RETURN ONLY $ SCAN($$BADOP,BADOP,BADOP,BADOP)$ GO SCO$

DEF.. ENTER GETAG $

IFN2($$SKPSET)$ M(12)= 2 $ SKPSET.. $M(14)= ( DEFEXPR( 0) )$ GO SCO$

BADOP.. RETWR(ILLGL())$ ORG.. N= DEFEXPR(0) $ 1432. IF NOT ABSL $ ( ABSL=VABS=1$ M(13)=N $ GO SCO )$

IF NOT VRBSW$ AND (M(13) EQL LOCNO) $ BEGIN COMMENT THIS IS AN INITIAL ORG, IT SUPERCEDES ORG 400 ASSUMPTION AND MAY HAVE ANY VALUE $ LOCNO = N $

M(13) = BTCLR( ''377'',N) $ END $
767... IF N LSS M(I3) $ GO BADOP$
776... LSTWRD = 0 $ N=N - M(I3) $ GO RPT1 $
1005... DIT.. N = DEFFEXPR(O) $
1010... RPT1... REPEAT N $ WRITE (LSTWRD) $ GO SCO $  
1025... VRB.. I3I = LOCN(I) $ VRBSW = I $  
1032... UPER=0$  
1033... ABSL = VABS $ GO BAS $  
1036... COMMA... ENTER GETAG $ M(I4)=(M(I3)-.UPPER)$ $  
1043... GO SCO$  
1044... GO E00... COMMENT A PASS NOW COMPLETED $  
1045... VARB = LOCN(O)$  
1051... I3=LOCN(I)-I$  
1056... LOCN(O)=LOCNO $ LOCN(1)=VARB $  
1065... IFN2 ( $ $ E001 ) $ COMMENT CK SYMBOL DEFS AFTER PASS 2 ONLY $  
1075... INCR(SCLR(8,LOCNO),BLK(7))$  
1077... I7=""3377""$  
1101... I1 = ""3400""$  
1103... E002.. IF I1 EQL ""3777"" $ GO EP3$  
1110... SWITCH M(I11) , (E003 , E002 , E004 ) $  
1130... GO E002$  
1131... E003..  
1131... UNDTAG = I1$  
1154... GO EP3$  
1135... E004.. INCR( VARB , M(I1-256)) $ GO E002 $  
1145... E001.. IF PASS EQL 3$ BEGIN  
1150... EP3..  
1152... PASS=PASS+1$  
1156... BLK(5) = ""5200""$  
1130... GO E002$  
1165... END$  
1165... EP4.. COMMENT SUCCESSFUL COMPLETION $  
1170... IF I7 NEQ ""3377"" $ TAPE(""704","",1,7)$  
1176... I1 = PROGNUM$  
1200... I2=LOCNO$  
1202... RDC(1,0,1,""230"" )$  
1204... DO ""6400""$  
1205... COMMENT EXECUTE A ROUTINE IN QTR 1 WHICH MOVES ASSEMBLED PROGRAM TO ITS DESIRED STACK LOCATIONS$  
1205...  
1205... BLINK.. I6I = BLNKS(O$  
1207... I4=I6$ GO BL1$  
1212... BL2..  
1217... M(I6I)=M(I1)+I4-I6-1$  
1222... BL1.. SCAN($$BL2,BADOP,BADOP,BADOP,BADOP)$  
1227... COMMENT NOW PROCESS NUMERIC PART OF BLINK TEXT $  
1227...  
1227... ARRAY FXP(0..3)=(0)$  
1227... FXP(1)=M(I3)$  
1232... BLOCN = READF(0)$  
1236... FXP(2)=LOCN(I)-READF(0)$  
1245... REPEAT BLOCN $  
1253... BEGIN  
1255... RELOC= READO)$  
1257... WORD = READF(0)$  
1262... FXP(3)=BLNKS(BTCLR(""7000"",WORD))$  
1274... WRITE(WORD+FXP(RELOC))$  
1307... END $
1310... VRBSW = 1 $ I3I = LOCN(1)$
1315... NXTBLW.. VRBADDR=READF(0)+FXP(2)$
1323... UNTIL VRBADDR EQL M(I3) $ WRITE(0)$
1333... IF VRBADDR EQL FXP(2) $ GO SCMO$
1341... WRITE(READF(0)) $ GO NXTBLW$
1346...
1346...
1347... FINISH$
COMMENT THIS IS THE LOSS MONITOR $ 

EXTERNAL PROCEDURE INCHAR( DMY )$

EXTERNAL PROCEDURE TYPEIHALF WORD CODE$ 

EXTERNAL PROCEDURE TYPEN ( NUMBER )$

ARY DOTS (2)= ("3636",7304"), 
PROG (4)= ("7334", "PROGS"), 
NOSHDR (8)= ("7334", "NO SUCH PROGS"), 
BADTXT (8)= ("7334", "NO SUCH TEXTS"), 
PARAMS (4)= ("PARAMS"), 
DEL (3)= ("3434", "XX"), 
BKUN (7)= ("7334", "BOOK UNDEF"), 
XECU (3)= ("7334", "RUN"), 
IHEDI (15)= ("7373", "NAME/LENGTH/BLOCK NO.", "73", "$$"), 
NWRMSG (3)= ("7334", 3673", "$$") , 
EMSG (4)= ("7334", "ERRORS")$

GLOBAL SGN$ 
GLOBAL QUOTE $ 
GLOBAL EMSG$ 
GLOBAL GETLINE, VAL $ 
GLOBAL DEL,NXTCHAR $ 

PROCEDURE BTWN (X,Y $$$ TRU )$ 
11... BEGIN 
24... IF (X LEQ M(I4)) AND ( M(I4) LEQ Y) $ GO TRU $ 
27... END BTWN ()$ 
33... PROCEDURE WRITE ($LOC)$ 
42... BEGIN 
42... ILL = LOC $ 
44... NEXT.. 
50... TYPE (MH(11))$ 
50... IF MH (11) NEQ 'S' $ GO NEXT $ 
55... END WRITE ()$ 
55... 
61... SUBROUTINE GETLINE$ 
65... BEGIN 
65... GETTOP.. I4 = "3300" $ 
67... TYPE("73")$ 
71... GETNXT... M(I4) = INCHAR (O)$ 
74... BTWN (O,"73" $$$ GETCONT )$ 
100... WRITE ($DEL(1))$ 
102... GO GETTOP$ 
103... GETCONT.. BTWN (O,"72" $$ GETNXT )$ 
107... RETURN $ 
110... END GETLINE$ 
110... 
110... SUBROUTINE NUTCHAR$ 
114... BEGIN 
114... IF M(I4) EQL "73"$ 
117... BEGIN 
117... ENTER GETLINE$
PROCEDURE SCAN ( $$ IDENT , NMBR , STR , SEMIC , QUEST , SLASH ) $ 
BEGIN
  SGN = 1$ 
  GLOBAL TOP$ 
  SCTOP..$ 
  BTWN ('-' , '-' $$MINUS )$ 
  BTWN ( 'A' , 'Z' $$ CHAR )$ 
  BTWN ( '0' , '7' $$ DIG )$ 
  BTWN ( 5 , 5 $$ STR1 )$ 
  BTWN ( '"33'" , '"33'" $$ SEM1 )$ 
  BTWN ( '/' , '/' $$ SLSH1 )$ 
  BTWN ( '"37'" , '"37'" $$ QUEST1 )$ 
  BTWN ( '"73'" , '"73'" $$SCSKP)$ 
  CHAR..$ 
  15=14$ 
  ENTER NXTCHAR$ 
  VAL = 64M(15)+M(14)$ 
  CHAR2. $ 
  BTWN ('A' , 'Z' $$ CHAR )$ 
  BTWN ( '0' , '9' $$ CHAR )$ 
  GO IDENT$ 
  CHAR2..$ 
  15=14$ 
  ENTER NXTCHAR$ 
  GO CHAR2$ 
  DIG.. $ 
  VAL = 0$ 
  DIG2.. $ 
  BTWN ( '"0'" , '"7'" $$ DIG )$ 
  GO NMBR$ 
  DIG1. $ 
  VAL = 8VAL + ABS (M(14) - '0').SGN$ 
  ENTER NXTCHAR$ 
  GO DIG2$ 
  STR1. $ 
  ENTER NXTCHAR$ 
  BTWN ( '"' , '"' $$ STR )$ 
  BTWN ( '"73'" , '"73'" $$ STR )$ 
  QUOTE = M(14)$ 
  GO STR$ 
  SEM1.. $ 
  ENTER NXTCHAR$ 
  GO SEMIC$ 
  SLSH1..$ 
  ENTER NXTCHAR$ GO SLASH$ 
  QUEST1..$ 
  ENTER NXTCHAR$ GO QUEST$ 
END SCAN()$ 

PROCEDURE ADD(ITEM,CONTROL)$ 
BEGIN
  M(121) = CONTROL$ 
  M(121) = ITEM$ 
END ADD()$
PROCEDURE SKPDOL( $$ FINAL )$

IF MH(I2) NEQ '$' $  
BEGIN 12 = 12 + '4000' $ GO L1 END $  
IF MH(I2I) EQL '$' $ GO FINAL $  
IF MH(I2) EQL '73' $ I2 = I2 + '4000' $  
END SKPDOL() $  

 TOP  
WRITE ( $PROG(1) ) $  
WRITE ( $DOTS(1) ) $  
I4 = '3300' $ M(I4)='73' $  
OVER=O$  
SCAN ( $$ PNAM , PNUM , ERR1 , ERR1 , TYPRS , ERR1 ) $  

PNAM..  
COMMENT USER HAS TYPED IN A PROGRAM NAME, FIND STACK NUMBER (N) AND GIVE HIM A DESCRIPTIVE MESSAGE $  
I2 = '2401' $ N = 0$  
PNAM. .. I1 = I5 $  
N = N + SGN$ COMMENT ALLOW NEGATIVE PROGRAM NAMES$  
PNAM2..  
IF MH(I2) EQL '$' $  
EITHER IF I1 EQL I4 $  
BEGIN  
SKPDOL($$ PNAM3) $  
BEGIN  
PNAM3.. I6 = I2 $ GO GETPS $  
END $  
END $  
OTHERWISE $ GO PNAM4 $  
IF MH(I2) EQL M(I1)$  
BEGIN  
I1=I1+1$ I2=I2+'4000' $  
GO PNAM2 $  
ENDS $  
PNAM4.. SKPDOL($$ PNAM5) $  
SKPDOL($$ PNAM5) $ GO PNAM1 $  

PNAM5.. WRITE($ NOSHPR(1) ) $ GO TOP $  

PNUM.. COMMENT OVERLAY AN UN-DESCRIBED PROGRAM $  
N = VAL $  
I6I = PARAMS(1) $  
GO GETPS $  

ERR1.  
WRITE($EMSG(1) )$  
GO TOP $  
TYPRS..  
COMMENT LIST ALL PROGRAMS $  
I2 = '2401' $  
TYPRS2..  
IF SNS 1 $ GO TOP $ COMMENT AVOID EXTRA OUTPUT $  
TYPE(''73'') $  
WRITE ($ M(I2) )$  

138.
540... SKPDOL($$ TOP) $ SKPDOL($$ TOP) $ 541... GO TYPIS2 $ 542... 543... GETPS.. 544... COMMENT GET PARAMETERS TO PROGRAM AND PUT THEM ON BUFFER $ 545... TYPE(''73'')$ 546... WRITE($M(16))$ 547... WRITE( $DOTS(1) )$ 548... BOOK = -1$ 549... I2 = POINTER + '4000' $ 550... 551... NXTTIM.. 552... I4='3300' $ 553... SCAN( $$ TNAM,NM, ST, SM, TYPIX, SLSH) $ 554... 555... NM.. 556... ADD(VAL,''3776'') $ 557... GO NXTITM $ 558... 559... ST.. 560... I3=I2=I2+1$ 561... 562... ENTER NXTCHAR $ 563... I5= I5 - I $ 564... 565... IF MH(I17) EQL 0$ 566... BEGIN 567... WRITE($BADTXT(1))$ COMMENT NOT IN THIS INDEX $ 568... GO NXTITM $ 569... 570... I7=I7+ROTR(1,MH(I7)+2)$ 571... REPEAT MH(I7) $ 572... IF I1 NEQ I4-1$ GO NOFND $ 573... IF I1 NEQ I4-1$ GO NOFND $ 574... L = MH(I17) $ 575... ADD(MH(I17)+BOOK,''3776'')$ 576... ADD( L , ''3776'') $ 577... GO NXTITM $ 578... NOFND.. I7=I7N $ GO TNAM1 $ 579... 580... SLSH.. 581... SCAN ( $$ ERR2, BKN, ERR2, ERR2, ERR2, ERR2 )$
754.. BKN.. COMMENT BOOK DEFINITION$
754.. BOOK=64BTCLR("7770",VAL)$
762.. RDC(0,1,7,BOOK)$
770.. GO NXTITEM$
770..
771.. ERR2.. WRITE($EMSG(1))$
773.. GO GETPS$
773..
773..
774.. TYPIX.. COMMENT TYPE CURRENT INDEX$
776.. IF BOOK LSS 0$ GO BKUNERR$
1002.. WRITE($XHED(1))$
1004.. i7="7377"$
1006.. UNTIL MH(I7) EQL 0$
1007.. BEGIN
1011.. IF SNS 1$ GO GETPS$
1013.. TYPE("73")$
1015.. REPEAT MH(I7)$ TYPE(MH(I7))$
1027.. TYPE("")$
1031.. TYPEN(MH(I7))$
1035.. TYPE("")$
1037.. TYPEN(MH(I7))$
1043.. ENDS
1044.. TYPE("73")$
1046.. GO GETPS$
1046..
1046..
1047..
1047..
1047.. COMMENT SEMI-OLON ENCOUNTERED,$
1047.. CAP RECORD, OVERLAY PROGRAM, THEN TYPE OUT BUFFER .$
1047.. COMMENT ZEROS FLAG END OF PARAM LIST IN CASE VRBL LNGTH$
1052.. ADD(0,"3776")$
1052.. ADD(0,"3776")$
1055.. ADD(0,"3776")$
1060.. ADD(0,"3776")$
1063.. M(I2)=POINTER$
1066.. POINTER = BTCLR("4000", I2)$
1073.. WRITE($XECU(1))$ WRITE($DOTS(1))$
1077..
1077.. IF N NEQ 0$
1101.. OVERLAY N$
1106.. TYPE("73")$
1110..
1110.. COMMENT TYPE BUFFER$
1110.. I3 = POINTER$
1112.. NEWR.. I2 = M(I3)$
1114.. WRITE($NWRMSG(1))$
1116.. IF I2 EQL 0$
1120.. BEGIN POINTER = I3$ GO TOP END$
1126.. NEWITEM..
1126.. IF I2 EQL I3-1$
1133.. BEGIN
1135.. I3=M(I3)$
1137.. GO NEWR$
1140.. ENDS
1140.. TYPE("73")$
1142.. IF M(I2) LSS 0$ GO ALPH$
1147.. REPEAT -BTSET("7000",M(I2))$
BEGIN COMMENT TYPE NUMBERS $  
TYPEN(M(I2I))$ TYPE(' ')$  
END$  
GO NEWITEM $  
ALPH.$  
I1 = I2 + "4000"$  
I2= I2 - M(I2) $  
REPEAT -2.M(I1) $  
BEGIN IF MH(I1I) EQL "74"$ EXIT NEWITEM $  
TYPE( MH(I1)) $  
END $  
GO NEWITEM $  
FINISH$
<table>
<thead>
<tr>
<th>CODE</th>
<th>1224 7674</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>16033</td>
</tr>
<tr>
<td>0010</td>
<td>01000</td>
</tr>
<tr>
<td>0020</td>
<td>01000</td>
</tr>
<tr>
<td>0030</td>
<td>01037</td>
</tr>
<tr>
<td>0040</td>
<td>00000</td>
</tr>
<tr>
<td>0050</td>
<td>01212</td>
</tr>
<tr>
<td>0060</td>
<td>06000</td>
</tr>
<tr>
<td>0070</td>
<td>00073</td>
</tr>
<tr>
<td>0100</td>
<td>16034</td>
</tr>
<tr>
<td>0110</td>
<td>16131</td>
</tr>
<tr>
<td>0120</td>
<td>16124</td>
</tr>
<tr>
<td>0130</td>
<td>00000</td>
</tr>
<tr>
<td>0140</td>
<td>12125</td>
</tr>
<tr>
<td>0150</td>
<td>00041</td>
</tr>
<tr>
<td>0160</td>
<td>00005</td>
</tr>
<tr>
<td>0170</td>
<td>00017</td>
</tr>
<tr>
<td>0200</td>
<td>00004</td>
</tr>
<tr>
<td>0210</td>
<td>27702</td>
</tr>
<tr>
<td>0220</td>
<td>00004</td>
</tr>
<tr>
<td>0230</td>
<td>00041</td>
</tr>
<tr>
<td>0240</td>
<td>16111</td>
</tr>
<tr>
<td>0250</td>
<td>00000</td>
</tr>
<tr>
<td>0260</td>
<td>01104</td>
</tr>
<tr>
<td>0270</td>
<td>16244</td>
</tr>
<tr>
<td>0280</td>
<td>00073</td>
</tr>
<tr>
<td>0300</td>
<td>16111</td>
</tr>
<tr>
<td>0310</td>
<td>16111</td>
</tr>
<tr>
<td>0320</td>
<td>14250</td>
</tr>
<tr>
<td>0330</td>
<td>14311</td>
</tr>
<tr>
<td>0340</td>
<td>00000</td>
</tr>
<tr>
<td>0350</td>
<td>16410</td>
</tr>
<tr>
<td>0360</td>
<td>12052</td>
</tr>
<tr>
<td>0370</td>
<td>01322</td>
</tr>
<tr>
<td>0380</td>
<td>01020</td>
</tr>
<tr>
<td>0410</td>
<td>16034</td>
</tr>
<tr>
<td>0420</td>
<td>01044</td>
</tr>
<tr>
<td>0430</td>
<td>16522</td>
</tr>
<tr>
<td>0440</td>
<td>00041</td>
</tr>
<tr>
<td>0450</td>
<td>00540</td>
</tr>
<tr>
<td>0460</td>
<td>16351</td>
</tr>
<tr>
<td>0470</td>
<td>00117</td>
</tr>
<tr>
<td>0500</td>
<td>12401</td>
</tr>
</tbody>
</table>

142.
<table>
<thead>
<tr>
<th>0510/</th>
<th>16440</th>
<th>16034</th>
<th>27700</th>
<th>16410</th>
<th>01000</th>
<th>10226</th>
<th>14437</th>
<th>00066</th>
</tr>
</thead>
<tbody>
<tr>
<td>0520/</td>
<td>27745</td>
<td>16545</td>
<td>16034</td>
<td>27702</td>
<td>16410</td>
<td>00062</td>
<td>02401</td>
<td>00461</td>
</tr>
<tr>
<td>0530/</td>
<td>16410</td>
<td>36002</td>
<td>00073</td>
<td>01000</td>
<td>00002</td>
<td>14537</td>
<td>16034</td>
<td>00000</td>
</tr>
<tr>
<td>0540/</td>
<td>16351</td>
<td>16410</td>
<td>16351</td>
<td>16410</td>
<td>16527</td>
<td>36002</td>
<td>00073</td>
<td>01000</td>
</tr>
<tr>
<td>0550/</td>
<td>0006</td>
<td>14555</td>
<td>16034</td>
<td>00000</td>
<td>16034</td>
<td>27775</td>
<td>01020</td>
<td>07776</td>
</tr>
<tr>
<td>0560/</td>
<td>01060</td>
<td>00000</td>
<td>01020</td>
<td>04000</td>
<td>02155</td>
<td>04002</td>
<td>00064</td>
<td>03300</td>
</tr>
<tr>
<td>0570/</td>
<td>1632</td>
<td>16644</td>
<td>16577</td>
<td>16606</td>
<td>17047</td>
<td>16774</td>
<td>16745</td>
<td>01000</td>
</tr>
<tr>
<td>0600/</td>
<td>10226</td>
<td>16033</td>
<td>00000</td>
<td>03776</td>
<td>15666</td>
<td>01020</td>
<td>00001</td>
<td></td>
</tr>
<tr>
<td>0610/</td>
<td>02002</td>
<td>04002</td>
<td>00043</td>
<td>00002</td>
<td>16111</td>
<td>01000</td>
<td>10304</td>
<td>00017</td>
</tr>
<tr>
<td>0620/</td>
<td>01104</td>
<td>00450</td>
<td>16641</td>
<td>16111</td>
<td>01020</td>
<td>00074</td>
<td>01362</td>
<td>01000</td>
</tr>
<tr>
<td>0630/</td>
<td>00002</td>
<td>01620</td>
<td>04000</td>
<td>04002</td>
<td>02002</td>
<td>00017</td>
<td>02003</td>
<td>01043</td>
</tr>
<tr>
<td>0640/</td>
<td>16566</td>
<td>01004</td>
<td>01362</td>
<td>16614</td>
<td>01000</td>
<td>10561</td>
<td>00450</td>
<td>00471</td>
</tr>
<tr>
<td>0650/</td>
<td>16654</td>
<td>16034</td>
<td>27733</td>
<td>16565</td>
<td>00067</td>
<td>07377</td>
<td>01020</td>
<td>07776</td>
</tr>
<tr>
<td>0660/</td>
<td>02005</td>
<td>04001</td>
<td>01327</td>
<td>00450</td>
<td>16670</td>
<td>16034</td>
<td>27751</td>
<td>16566</td>
</tr>
<tr>
<td>0670/</td>
<td>01307</td>
<td>12630</td>
<td>00301</td>
<td>02007</td>
<td>01060</td>
<td>00000</td>
<td>01307</td>
<td>14702</td>
</tr>
<tr>
<td>0700/</td>
<td>06123</td>
<td>16713</td>
<td>00000</td>
<td>01327</td>
<td>00017</td>
<td>01121</td>
<td>00470</td>
<td>16712</td>
</tr>
<tr>
<td>0710/</td>
<td>06046</td>
<td>16742</td>
<td>06146</td>
<td>01020</td>
<td>07776</td>
<td>02004</td>
<td>00017</td>
<td>02001</td>
</tr>
<tr>
<td>0720/</td>
<td>00450</td>
<td>16742</td>
<td>01327</td>
<td>01060</td>
<td>00000</td>
<td>01327</td>
<td>12561</td>
<td>14731</td>
</tr>
<tr>
<td>0730/</td>
<td>16333</td>
<td>00000</td>
<td>03776</td>
<td>01000</td>
<td>10724</td>
<td>14737</td>
<td>16333</td>
<td>00000</td>
</tr>
<tr>
<td>0740/</td>
<td>03776</td>
<td>16566</td>
<td>0047</td>
<td>10675</td>
<td>16556</td>
<td>16132</td>
<td>16771</td>
<td>16754</td>
</tr>
<tr>
<td>0750/</td>
<td>16771</td>
<td>16771</td>
<td>16771</td>
<td>01000</td>
<td>10226</td>
<td>01560</td>
<td>07770</td>
<td></td>
</tr>
<tr>
<td>0760/</td>
<td>00246</td>
<td>15661</td>
<td>01120</td>
<td>07000</td>
<td>12561</td>
<td>14767</td>
<td>00710</td>
<td>00000</td>
</tr>
<tr>
<td>0770/</td>
<td>16566</td>
<td>16034</td>
<td>27702</td>
<td>16545</td>
<td>01000</td>
<td>10561</td>
<td>00470</td>
<td>17002</td>
</tr>
<tr>
<td>1000/</td>
<td>00451</td>
<td>16551</td>
<td>16034</td>
<td>27711</td>
<td>00067</td>
<td>07377</td>
<td>01327</td>
<td>00470</td>
</tr>
<tr>
<td>1010/</td>
<td>17044</td>
<td>00461</td>
<td>16545</td>
<td>36002</td>
<td>00073</td>
<td>01307</td>
<td>15021</td>
<td>06123</td>
</tr>
<tr>
<td>1020/</td>
<td>17027</td>
<td>00000</td>
<td>01327</td>
<td>15025</td>
<td>36002</td>
<td>00000</td>
<td>06146</td>
<td>36002</td>
</tr>
<tr>
<td>1030/</td>
<td>00000</td>
<td>01327</td>
<td>15034</td>
<td>36003</td>
<td>00000</td>
<td>36002</td>
<td>00000</td>
<td>01327</td>
</tr>
<tr>
<td>1040/</td>
<td>15042</td>
<td>36003</td>
<td>00000</td>
<td>17006</td>
<td>36002</td>
<td>00073</td>
<td>16545</td>
<td>16333</td>
</tr>
<tr>
<td>1050/</td>
<td>00000</td>
<td>03776</td>
<td>16333</td>
<td>00000</td>
<td>03776</td>
<td>16333</td>
<td>00000</td>
<td>03776</td>
</tr>
<tr>
<td>1060/</td>
<td>16333</td>
<td>00000</td>
<td>03776</td>
<td>01000</td>
<td>01555</td>
<td>01062</td>
<td>01000</td>
<td>00002</td>
</tr>
<tr>
<td>1070/</td>
<td>01560</td>
<td>04000</td>
<td>04155</td>
<td>16034</td>
<td>27730</td>
<td>16034</td>
<td>27775</td>
<td>01000</td>
</tr>
<tr>
<td>1100/</td>
<td>01043</td>
<td>00470</td>
<td>17106</td>
<td>01000</td>
<td>10437</td>
<td>06343</td>
<td>36002</td>
<td>00073</td>
</tr>
<tr>
<td>1110/</td>
<td>00043</td>
<td>00155</td>
<td>01003</td>
<td>04002</td>
<td>16034</td>
<td>27706</td>
<td>01000</td>
<td>00002</td>
</tr>
<tr>
<td>1120/</td>
<td>00450</td>
<td>17126</td>
<td>01000</td>
<td>00003</td>
<td>04155</td>
<td>16410</td>
<td>01200</td>
<td>07776</td>
</tr>
<tr>
<td>1130/</td>
<td>02003</td>
<td>00117</td>
<td>02002</td>
<td>00450</td>
<td>17140</td>
<td>01003</td>
<td>04003</td>
<td>17112</td>
</tr>
<tr>
<td>1140/</td>
<td>36002</td>
<td>00073</td>
<td>01022</td>
<td>00470</td>
<td>17147</td>
<td>00451</td>
<td>17167</td>
<td>01002</td>
</tr>
<tr>
<td>1150/</td>
<td>01620</td>
<td>07000</td>
<td>00117</td>
<td>15156</td>
<td>06123</td>
<td>17166</td>
<td>00000</td>
<td>01022</td>
</tr>
<tr>
<td>1160/</td>
<td>15162</td>
<td>36003</td>
<td>00000</td>
<td>36002</td>
<td>00000</td>
<td>06146</td>
<td>17126</td>
<td>01020</td>
</tr>
<tr>
<td>1170/</td>
<td>04000</td>
<td>02002</td>
<td>04001</td>
<td>01102</td>
<td>00117</td>
<td>02002</td>
<td>04002</td>
<td>01101</td>
</tr>
<tr>
<td>1200/</td>
<td>00241</td>
<td>00117</td>
<td>15203</td>
<td>06123</td>
<td>17222</td>
<td>00000</td>
<td>01321</td>
<td>01120</td>
</tr>
<tr>
<td>1210/</td>
<td>07003</td>
<td>00450</td>
<td>17215</td>
<td>06046</td>
<td>17126</td>
<td>01301</td>
<td>15220</td>
<td>36002</td>
</tr>
<tr>
<td>1220/</td>
<td>00000</td>
<td>06146</td>
<td>17126</td>
<td>06336</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
/*VARB
/7702 7334 7703 4562 7704 6257 7705 6204 7706 7334 7707 3673 7710 0404 7711 7373 7712 5641
/7713 5545 7714 1754 7715 4556 7716 4764 7717 5017 7720 4254 7721 5743 7722 5300 7723 5657
/7724 1600 7725 0073 7726 0404 7730 7334 7731 6265 7732 5604 7733 7334 7734 4257 7735 5753
/7736 0065 7737 5644 7740 4546 7741 0400 7742 3434 7743 7070 7744 0400 7745 6041 7746 6241
/7747 5563 7750 0400 7751 7334 7752 5657 7753 0063 7754 6543 7755 5000 7756 6445 7757 7064
/7760 0400 7761 7334 7762 5657 7763 0063 7764 6543 7765 5000 7766 6062 7767 5747 7770 0400
/7771 7334 7772 6062 7773 5747 7774 0400 7775 3636 7776 7304
/*FINISH
COMPILED PROGRAM ENDS AT 1223
PROGRAM VARIABLES BEGIN AT 7674
### Symbols in Main Program

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DEF</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Simple Variables

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DEF</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10561</td>
<td>LO INT</td>
<td>BOOK</td>
</tr>
<tr>
<td>24</td>
<td>GL INT</td>
<td>I41</td>
</tr>
<tr>
<td>10724</td>
<td>LO INT</td>
<td>L</td>
</tr>
</tbody>
</table>

#### Arrays

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DEF</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>27750</td>
<td>LO INT</td>
<td>BADTXT</td>
</tr>
<tr>
<td>27710</td>
<td>LO INT</td>
<td>IXHED</td>
</tr>
<tr>
<td>27705</td>
<td>LO INT</td>
<td>NWRMSG</td>
</tr>
</tbody>
</table>

#### Procedures and Functions

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DEF</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10333</td>
<td>GL INT</td>
<td>ADD</td>
</tr>
<tr>
<td>164</td>
<td>GL INT</td>
<td>PUT</td>
</tr>
<tr>
<td>30002</td>
<td>GL INT</td>
<td>TYPE</td>
</tr>
</tbody>
</table>

#### Labels

<table>
<thead>
<tr>
<th>ADDR</th>
<th>DEF</th>
<th>TYPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>11167</td>
<td>LO ALPH</td>
<td>NEWR</td>
</tr>
<tr>
<td>10577</td>
<td>LO NM</td>
<td>PNUM</td>
</tr>
<tr>
<td>10651</td>
<td>LO BKN</td>
<td>SLSSH</td>
</tr>
<tr>
<td>10522</td>
<td>LO ERR1</td>
<td>SML</td>
</tr>
<tr>
<td>10772</td>
<td>LO ERR2</td>
<td>SM</td>
</tr>
<tr>
<td>10103</td>
<td>LO GETCONT</td>
<td>ST</td>
</tr>
<tr>
<td>10062</td>
<td>LO GETLINE</td>
<td>ST1</td>
</tr>
<tr>
<td>10071</td>
<td>LO GETNX</td>
<td>TNAME</td>
</tr>
<tr>
<td>10545</td>
<td>LO GETPS</td>
<td>TNAME1</td>
</tr>
<tr>
<td>10065</td>
<td>LO GETTOP</td>
<td>TOP</td>
</tr>
<tr>
<td>11126</td>
<td>LO NEWITEM</td>
<td>TYPJOB</td>
</tr>
</tbody>
</table>

---

145.
SYMBOLES IN PROCEDURE/FUNCTION 'BTWN'

10006 LO INT X
NA TRU

10007 LO INT Y

SYMBOLES IN PROCEDURE/FUNCTION 'WRITE'

SYMBOLES IN PROCEDURE/FUNCTION 'SCAN'

SYMBOLES IN PROCEDURE/FUNCTION 'ADD'

SYMBOLES IN PROCEDURE/FUNCTION 'SKPDOL'

10357 LO L1

EXECUTION, NO ERRORS FOUND
STANFORD BLINK COMPILER — VERSION OF 20 MAY 65

COMMENTS THIS IS THE QUESTION AND ANSWER PROGRAM$

**--SPACE**

**--**

EXTERNAL PROCEDURE INCHAR(DMY)$

EXTERNAL PROCEDURE TYPE(HALF WORD CODE)$

EXTERNAL PROCEDURE TYPEN( NUMBER )$

ARRAY DOTS(2)=('"3636', '7304''),

PROG(4) = ('"7334', 'PROGS'),

NOHDR(8) = ('"7334', 'NO SUCH PROGS'),

BADTXT(8) = ('"7334', 'NO SUCH TEXTS'),

PARAMS(4) = ('PARAMS'),

DEL(3) = ('"3434', 'XX$'),

BKUN (7) = ('"7334', 'BOOK UNDEF$'),

EXECU (3) = ('"7334', 'RUN$'),

IXHED(15) = ('"7373', 'NAME/LENGTH/BLOCK NO.', '73', 'S$'),

NMRMSG(3) = ('"7334', '3673', 'S$'),

EMSG(4) = ('"7334', 'ERROR$')$

GLOBAL QUOTE$

GLOBAL SGN$

GLOBAL EMSG[]$

GLOBAL GETLINE, VAL

GLOBAL DEL,NXTCHAR$

PROCEDURE BTWN(X,Y $$ TRU)$

BEGIN

IF (X LEQ MH(I4)) AND (MH(I4) LEQ Y)$ GO TRU$ END BTWN()$

BEGIN

III = LOC$

NEXT.. TYPE(MH(II))$

IF MH(III) NEQ '$' GO NEXT$

END WRITE()$

SUBROUTINE GETLINE$

BEGIN

GETTOP.. I4 = '"3340'"

TYPE('"73''')$

GETNXT.. MH(I4) = INCHAR(0)$

BTWN(0,'"73'' $ GETCONT )$

WRITE($DEL(1))$

GO GETTOP$

GETCONT.. BTWN(0,'"72'' $ GETNXT )$

RETURN$

END GETLINE$

SUBROUTINE NXTCHAR$

BEGIN

IF MH(I4) EQL '"73''$
BEGIN
ENTER GETLINE $
I4 = '3340'$
END $
I4 = I4 + '4000'$
RETURN END NXTCHAR $

PROCEDURE SCAN ( $$ IDENT , NMBR , STR , SEMIC , QUEST , SLASH )$
BEGIN
SGN = 1$
GLOBAL TOP$

BEGIN
SGN = 1$
GLOBAL TOP$
BTWN('-','-' $$MINUS)$
BTWN ( 'A' , 'Z' $$ CHAR )$
BTWN ( '0' , '7' $$ DIG )$
BTWN ( 5 , 5 $$ STR1 )$
BTWN ( '33' , '33' $$ SEM1 )$
BTWN ( '/' , '/' $$ SLSH1 )$
BTWN( '37' , '37' $$ QUEST1 )$
BTWN ( '73' , '73' $$SCSKP)$
WRITE( $$EMSG1)$
GO TOP$
MINUS.. SGN : -1$

SCSKP.. ENTER NXTCHAR$
GO SCTOP$
CHAR.. I5=I4 $
ENTER NXTCHAR$
VAL = 64MH(I5) + MH(I4)$
CHAR1.. BTWN("A","Z" $$ CHAR2 )$
BTWN ( '0' , '9' $$ CHAR2 )$
GO IDENT$

GO IDENT$

CHAR2.. ENTER NXTCHAR$
GO CHAR1$
DIG.. VAL = 0$
DIG2.. BTWN ('0' , '7' $$ DIG1 )$
GO NMBR$
DIG1.. VAL = 8VAL + ABS (MH(I4) - '0').SGN$
ENTER NXTCHAR$
GO DIG2$
STR1.. ENTER NXTCHAR$
BTWN ( '.' , '.' $$ STR1 )$
BTWN ( '73' , '73' $$ STR1 )$
QUOTE = MH(I4)$
GO STR$
SEMI.. ENTER NXTCHAR$
GO SEMIC$
SLSH1.. ENTER NXTCHAR GO SLASH$
QUEST.. ENTER NXTCHAR GO QUEST$
END SCAN()$

PROCEDURE ADD(ITEM,CONTROL) $
BEGIN
M(I2I) = CONTROL$
PROCEDURE SKPDOL( $$ $ END $$ $)

BEGIN
  L1.. IF MH(I2) NEQ$ $ BEGIN I2 = 12 +$ $4000$$ $ GO L1 END $ $363.. IF MH(I2) EQL $ $ $$ $2$$ $ $ GO FINAL $ $372.. IF MH(I2) EQL$ $1$$ $ $ I2 = 12 +$ $4000$$ $ $405.. END SKPDOL()$$ $405.. $411.. TYPE($$`73'') $$ $413.. COMMENT TYPE TOP RECORDS $ $415.. I3 = POINTER $ $417.. NEWR. I2 = M(I3) $ $420.. WRITE($$NWRMSG(I1)$$ $423.. IF I2 EQL 0 $ $ BEGIN POINTER = I3 $ $ GO TOP $ $ END $ $431.. NEWMITEM.. $ $431.. IF I2 EQL I3-1$ $ BEGIN $ $440.. I3 = M(I3)$$ $ $442.. GO TOP1$ $443.. END$ $ $445.. TYPE($$`73'') $$ $452.. REPEAT -BSET($$`7000',M(I2)$$ $ $462.. BEGIN COMMENT TYPE NUMBERS $ $466.. TYPEN(M(I2))$ $ TYPE(' ')$ $470.. END$ $471.. GO NEWMITEM $ $471.. $ $472.. ALPH.. I1 = I2 +$ $4000$$ $ $476.. I2 = I2 - M(I2) $$ $502.. REPEAT -2*M(I1) $$ $511.. BEGIN IF MH(I1) EQL $ $74$$ $ $ EXIT NEWMITEM $ $520.. TYPE( MH(I1))$ $ $524.. END $ $525.. GO NEWMITEM $ $525.. $ $525.. $ $525.. $ $525.. $ $526.. TOP1.. POINTER = I3 $ $531.. $ $531.. $ $531.. $ $531.. GETPS.. $ $533.. COMMENT GET PARAMETERS TO PROGRAM AND PUT THEM ON BUFFER $ $533.. TYPE($$`73'')$$ $533.. WRITE($$DOTS(I1)$$ $535.. BOOK = -1$$ $541.. I2 = POINTER +$ $4000$$ $ $541.. $ $545.. NXTITM.. $ $547.. I4 = $$`3340'$$ $ MH(I4) = $$`73'$$ $ $552.. SCAN($$ $$TNAM,NM, ST, SM, TYPIX, SLSH)$$ $ $561..
561. NM
567. ADD(VAL, "3776")$
567. GO NXTITM$
567. 
570. ST
570. I3 = I2 = I2 + 1$
576. ST1
576. ENTER NXTCHAR$
577. IF MH(I4) EQL QUOTE$
602. BEGIN
602. ENTER NXTCHAR$
605. MH(I2I) = "74"$
610. I2 = RTSET( "4000", I2 )$
615. M(I3) = I3 - I2$
621. GO NXTITM$
622. END$
622. MH(I2I) = MH(I4)$
624. GO ST1$
624. 
625. TNAM.. COMMENT FIND THIS NAME IN CURRENT BOOK INDEX
625. AND PUT THE BLOCK NUMBER AND LENGTH ON BUFFER$
627. IF BOOK LSS 0$ (BKUNERR.. WRITE($BKUN(1))$ GO NXTITM )$
635. 17="7377"$
637. TNAM1.. 11 = 15 - "4000"$
643. IF MH(I7I) EQL 0$
644. BEGIN
650. WRITE($BADTXT(1))$ COMMENT NOT IN THIS INDEX$
650. GO NXTITM$
651. END$
651. I7N = I7 + ROTR(1, MH(I7)+2)$
657. REPEAT MH(I7)$
664. IF MH(I7I) NEQ MH(I7I) $ EXIT NOFND$
677. IF I1 NEQ I4 - "4000"$ GO NOFND$
706. L = MH(I7I)$
711. ADD(MH(I7I)+BOOK, "3776")$
717. ADD( L, "3776")$
725. GO NXTITM$
726. NOFND.. 17 = I7N$ GO TNAM1$
730. 
731. SLSH..
740. SCAN ( $$ ERR2, BKN, ERR2, ERR2, ERR2, ERR2 )$
740. BKN.. COMMENT BOOK DEFINITION$
740. BOOK = 64BTCLR("7770",VAL)$
746. RDC(0,1,7,BOOK)$
753. GO NXTITM$
753. 
754. ERR1.. 
754. ERR2.. 
756. WRITE($EMSG(1))$
756. GO GETPS$
756. 
757. TYPIX.. COMMENT TYPE CURRENT INDEX$
761. IF BOOK LSS 0$ GO BKUNERR$
765. WRITE($IXHED(1))$
767. 17="7377"$
UNTIL MH(I7I) EQL 0$
BEGIN
IF SNS 1$ GO GETPS $
TYPE(''73'')$
REPEAT MH(I7)$ TYPE(MH(I7I))$
TYPE('' '')$
TYPEN(MH(I7I))$
TYPE('' '')$
TYPEN(MH(I7I))$
END$
TYPE(''*73'')$
GO GETPS $
SM.. 
ADD(O,''3776'')$
ADD(O,''3776'')$
ADD(O,''3776'')$
ADD(O,''3776'')$
M(I2I)=POINTER $
POINTER = BTCLR(''4000'', I2) $
WRITE($XECU(I))$ WRITE($DOTS(1))$
FINISH$
--SPACE

COMMENT THIS PROGRAM DEFINES AND ERASES TEXTS.

ITS INPUT IS A SINGLE ALPHABETIC RECORD, THE OPERATIONS
EXPECTED ARE:

BOOK N ... THE N'TH BOOK IS CONSIDERED,
DEFINE NAME(N) ... THE TEXT 'NAME' IS DEFINED
WITH LENGTH N,

ERASE NAME.

STORAGE ALLOCATION IS AS FOLLOWS:
QUARTER 4.. STRINGS AND ARRAYS
QUARTER 5.. TEMP INDEX STORAGE
QUARTER 7.. CURRENT INDEX $

GLOBAL BADIX , FULL, BDOPR,AVAIL,COUNT,CHAR $
GLOBAL BADOPS$.

STRING
FULL(12)=('BOOK CAPACITY EXCEEDED'),
BDOPR(IO)=('ILLEGAL OPERATION')$
ARRAY AVAIL(O..66),CHAR(10) $

PROCEDURE GETEL($$ALPHA,NUMBER)$
7.. BEGIN
33.. FUNCTION BTWN(X,A,Y)= (X LEQ A) AND (A LEQ Y) $
33..
43.. GCH.. 46.. EITHER IF MH(16) EQL ''74''$ RETURN$
51.. OR IF BTWN('A',MH(16),''Z'')$
60.. BEGIN
64.. III=CHAR(1)$ I2=11$
66.. MH(III)=MH(16)$
70.. UNTIL NOT(BTWN('A',MH(16I),''Z'') OR BTWN('O',MH(16),'9'))$
106.. MH(III)=MH(16)$
113.. MH(I2)=2(11-I2)$
121.. MH(III)=0$
123.. GO ALPHA$
124.. OR IF BTWN('O',MH(16),'7')$
135.. BEGIN
140.. CHAR(1)=ABS(MH(16)-'0')$
141.. UNTIL NOT BTWN('O',MH(16I),'7')$
152.. CHAR(1)=CHAR(1)B+ABS(MH(16)-''0'')$
172.. GO NUMBERS$
172.. OTHERWISE$ INCR(''4000'',16)$$
177.. GO GCH$$
200.. END GETEL()$
209.. SUBROUTINE WRITE$
212.. BEGIN
214.. IF BOOK GEQ O$ WRC(O,1,7,BOOK) $
226.. RETURN END WRITE$
PROCEDURE ERITE(STR())$
BEGIN
STRING S(11)=('AT OPERATION NUMBER')$
LIST L(STR(),S(),COUNT)$
PUT($SL)$
OVERLAY 0$
END ERITE($)

PROCEDURE SEARCH($$FOUND)$$
BEGIN
I7=''3377''$
I5=''2377''$
REPEAT 256 $ M(I5)=M(I7)$
I7=''7377''$
GETEL($$SRCH!,BADOP)$ GO BADOP$
SRCH!..$
UNTIL MH(I7) EQL 0$
BEGIN
III=CHAR(1)$
MATCH=O$
I5=I7-(''1000''+''4000'')$
I2=I7+ROTR(1,MH(I7)+2)$
REPEAT MH(I7)<=IF MATCH EQL 0$
MATCH=MH(III)-MH(I7)$
I7=I2$
IF MATCH LSS 0$
BEGIN COMMENT NOT PRESENT$
I7=I5+''1000''$
RETURN$
ENDS
IF MATCH EQL 0$
IF MH(I5) EQL 0$ GO FOUNDS
COMMENT KEEP LOOKING$
END$
COMMENT NOT PRESENT$
I5=INCR(-''4000'',I7)-''1000''$
END SEARCH()$
SUBROUTINE FINIX$
BEGIN
UNTIL MH(I7) EQL 0$
BEGIN MH(I5)=MH(I7)$
REPEAT MH(I7)+2 $ MH(I5)=MH(I7)$
END$
UNTIL I5 EQL ''6777'' $ MH(I5)=0$
COMMENT NOW MOVE NEW INDEX TO QUARTER 7$
I7=''3377''$ I5=''2377''$
REPEAT 256$ $ M(I7)=M(I5)$
RETURN END FINIX$
BOOK = -1$ COUNT=0$
I6=M(POINTER)+2$ IF M(I6-1) GTR 0$ GO BADOP$
504... DINTER = M(POINTER) $
512...$ OPR.. COUNT=COUNT+1 $\$
520...$ GETE1($$ALPH,BADOP)$
523... N0M0.. ENTER WRITES
524... RETURN $
525...$ ALPH.. IF MH(I21) EQL 'B' $
530...$ BEGIN
535... GETE1($$BADOP,NWBK)$ GO BADOP $
536...$ NWBK.. ENTER WRITE $
537...$ BOOK=BTCLR('77770','CHAR(1))64 $
545...$ RDC(0,1,7, BOOK) $
552...$ I4=AVAIL(0)$ BASE=I4$
560...$ REPEAT 64$ M(I4I)=O$
566...$ I7='7377''$
570... UNTIL MH(I7I) EQL 0$
571... BEGIN
573... I7=I7+ROTR(I,MH(I7))$
577... SIZE=MH(I7I)$
602... I4=BASE+MH(I7I)-1$
606... REPEAT SIZE $ M(I4I)=1$
620... ENDS
621... GO OPR $
622... IF BOOK LSS O$ GO BADOP $
630... IF MH(I2) EQL 'D' $
633...$ BEGIN
637... SEARCH($$BADOP)$
637... I1I=CHAR(1)$ MH(I5I)=MH(I1)$
643... REPEAT MH(I1)$ MH(I5I)=MH(I1)$
653... GETE1($$BADOP,LGTH)$ GO BADOP $
657... LGTH.. MH(I5I)=CHAR(1)$
662... I4=BASE$
664... ZERS=0$
667...$ IF I4 EQL BASE + 63 $ ERITE(FULL())$
674... EITHER IF M(I4I) EQL 1 $ ZERS=0$
701... OTHERW1SE$ ZERS=ZERS+1$
715... IF ZERS NEQ MH(I5)$ GO ITER $
722... I4=I4-ZERS$
727... MH(I5I)=I4-BASE+1$
734... REPEAT ZERS $ M(I4I)=1$
740... ENTER FINIX $
747... GO OPR $
750... ENDS
750... IF MH(I2) EQL 'E'$
753... BEGIN $
753... SEARCH($$ERFND)$
757... BADOP.. ERITE(BDOPR(1))$
762... ERFND.. I4=BASE+MH(I7)-1$
766... REPEAT MH(I7-''4000''$ M(I4I)=0 $
1002... ENTER FINIX $
1003... GO OPR $
1004... END$
COMMENT THIS PROGRAM LOADS A BLINK TEXT

EXTERNAL PROCEDURE INREC (NSMINL, MAXV$ERRCAS)

EXTERNAL SUBROUTINE CONVRT

ARRAY BUFF(0..10)

GLOBAL B, BMX, ERR, SSAV, NL

GLOBAL SENDN, PAR$ S

GLOBAL BS$AVE

**--SPACE**

STRING

OK(6)=('BLINK LOADED'),

SNS1(2)=('SNS1'),

LREC(5)=('REC OFLOW'),

FULL(5)=('TXT OFLOW')

GLOBAL OK, SNS1, LREC, FULL

PROCEDURE QUIT( STR($)

10... BEGIN

17... LIST L(STR($)

20... STRING $1 (7)=('TAPE ERRORS')$ 

20... LIST L1(S1($)

26... M(''21''$=BSAVE$ 

32... IF ERR$ PUT($$L1$)

41... PUT($$L$)

43... OVERLAY 0$ 

45... END QUIT($$)

45...

46... PROCEDURE SEND(N$)

55... BEGIN

55... N=ABS(N$)

61... II=SSAV$

63... IF II EQL '6377''$ 

66... BEGIN

74... WRC(0,1,4,INCR(1,8))$

84... II=0$ II=-II$

97... IF B EQL BMX$ QUIT( FULL())$

107... END$

107... END SEND($$

127... SUBROUTINE SENDN$

133... BEGIN

133... REPEAT 2$ 

136... BEGIN

151... SEND(8(MH(I7))+MH(I7I)-(8'0'+'0'))$

151... I7=I7++'4000''$

155... END$
156... RETURN END SENDN$
157...
157... PROCEDURE TSTEOL($$L)$
166... BEGIN
166... ENTER SENDN$
167... IF MH(I7) EQL ' ' GO NL$
172... IF MH(I7) EQL ' '73''$ GO NL$
177... GO L$
200... END TSTEOL($)$
200...
204... WRC(0,0,1, ''170'')$
206... BSAVE=M(''21'')$ III=BUFF(0)$ M(''21'')=II$
216... SSAVE = ''6000''$
217... GET ($$PARS)$
222... IF BLK LEQ 0$
224... QUIT(FULL())$
233... B=BLK-1$ BMAX=B+LENGTH$
237...
243... ERR1 = 0$
246... INREC(I$M(''2401''),i$ERRCAS)$
253... CONV_.. ENTER CONVRT$
254... I6=''6400''$
256... CKD=0$
256...
261... IF SNS 1$ QUIT( SNS1())$
266... I7=I6$
270... IF MH(161) EQL ' '74''$ GO NR$
275... IF MH(I6) NEQ ' '73''$ GO NC$
301... IF MH(I71) NEQ ' '7$ GO NL$
304... IF MH(I71) NEQ ' '1''$ GO NL$
311... IF NOT BLFND$
313... BEGIN BLFND=(MH(I7+1) EQL 'B')$ GO NL$ END$
335... COMMENT TAPE ERRORS ONLY NOW SIGNIFICANT$
340... IF I5 EQL ' '3777''$ QUIT( LREC())$
345... IF NOT CKD$
347... BEGIN
354... CKD = 1$ IF ERR1$ ERR= 1$
362... ENDS
362... IF MH(I71) EQL ' '*''$ GO PSEUDOS$
367... IF VRBFND$
371... BEGIN
375... NXTV.. TSTEOL($$NXTV)$
375... ENDS
375... COMMENT CODE CASE, SKIP ADDRESS$
375... I7=I7+3$
401... NEXTC.. SEND(MH(I7)-'0')$
407... I7=I7+''4000''$
413... TSTEOL($$NEXTC)$
415... PSEUDOS...
415... EITHER IF MH(I71) EQL 'E'$
420... BEGIN COMMENT 'EXT' HAS APPEARED$
420... I7=I7+''4001''$
426... UNTIL MH(I71) EQL ' ' SEND(MH(I7))$
436... SEND(' ')$
440... ENDS

156.
OR IF MH(I7) EQL 'C'
BEGIN COMMENT 'CODE'
SEND("'73'"$) COMMENT TERMINATE SYMBOLIC PARTS
I7=I7+"4002"
ENTER SENDN
I7=I7+"4000"
ENTER SENDN
END$
OR IF MH(I7) EQL 'V'
BEGIN COMMENT 'VARB'
VRBFND=1
END$
OR IF MH(I7) EQL 'F'
BEGIN COMMENT 'FINISH'
SEND(0)$
SEND(0)$
WRC(0,1,4,INCR(1,B))$
RDC(0,1,4,BLK)$
MH("'2000'")=B-BLK+1$
MH("'6000'")=B$
WRC(0,1,4,BLK)$
QUIT(0)$
END$
OTHERWISE$ ERR=1$
GO NL$
ERRCAS.. ERR1 = 1$ GO CONV$
LIST PARS (BLK,LENGTH)$
FINISH$