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

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 thereafollowing becomes the delimiter of the string. All characters appearing after the first occurence of the delimiter, but before its second occurence, 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>>                          (monitor requests prog name
*     TYPER                            (user mis-types
      <NO SUCH PROG                    (monitor flags error
      <PROG>>                          (request repeated
*     TYPE                             (correction made
      <NO. OF FILES, FILES>>           (parameter request
*     3                                (first paramater
*     PROGA                            (file name entered
      <BOOK UNDEF                      (but book no. forgotten
*     /1                               (book number
*     PROGA                            (desired name repeated
*     PROGB                            (a second file
*     PROGC                            (a third file
      <NO SUCH FILE                    (PROGC not in book 1
*     /2                               (new book no.
*     PROGC                            (third file now correct
*     ;                                (end of params
      <RUN                             (execution begins
```

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.

basic program stack

stack location number	program name
1-7	(unused)
10	TYPE
11	BLINK
12	DISTAPE
13	DISPLAY
14	DEFINE
15	MONITOR
16	EDIT
17-22	(overlay stack)
23	QA
24	LASS
25	LASS (pass 2)
26-	(unused)

LOSS Character Codes.

CHARACTER	CODE
	00
!	01
"	02
#	03
$	04
%	05
&	06
'	07
(	10
)	11
*	12
+	13
,	14
-	15
.	16
/	17
0	20
1	21
2	22
3	23
4	24
5	25
6	26
7	27
8	30
9	31
:	32
;	33
<	34
=	35
>	36
?	37
@	40

CHARACTER	CODE
A	41
B	42
C	43
D	44
E	45
F	46
G	47
H	50
I	51
J	52
K	53
L	54
M	55
N	56
O	57
P	60
Q	61
R	62
S	63
T	64
U	65
V	66
W	67
X	70
Y	71
Z	72
(carr. ret.)	73
(end of string)	74
(or)	
(end of text)	

code derivation:

```
LDA
(teletype code)
GOM
ADA i
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>>                          (param request
*   $+ BOOK 6 DEFINE                    (begin a string
*   CHARLIE 7 +                         (carr. ret. allowed
*   ;
    <RUN>>
```

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$
*   ;
    <RUN>>
```

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
```

The DEFINE program scans its input string for occurences 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
χ	:	a carriage return
μ	:	an arbitrary character (constant within a single operation)
β	:	an arbitrary string of characters
s	:	a space

Command	Effect
nS or nχ	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.
nA (Ahead)	The line of attention is advanced n lines.
nB (Back)	The line of attention is moved back n lines.
A(B)	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).
Fμβμ (Find)	(where β does not contain μ) The text is searched (beginning at the line of attention) for an occurence of β. If an occurence is found, the line of attention is advanced so that β appears on the screen. If no occurence is found, attention is set at the end of the text.
,	The next occurence of β is sought. Thus the following sequence of commands would cause the fourth occurence of the word JAZZ to be displayed (or else the end-of-text symbol): F*JAZZ*,,,
nD (Delete)	n lines are deleted from the text, beginning with the line of attention. If n is absent, only the line of attention is deleted.
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.

S$\mu\beta_1\mu\beta_2\mu$ (Substitute) (where neither β_1 nor β_2 contain μ) The text, beginning at the line of attention, is searched for occurences of β_1; every such occurence is replaced in the text by β_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) SJMPJMP$
this operation counts the number of times the string JMP appears in the text.

(2) S.Q.LDA,0,JMP 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/RECIEVE/RECEIVE/
Here, a spelling error is corrected.

L	(Line)	The current line number is typed out.
nT	(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 <u>not</u> 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                      book specification
*   %"NAME1:9/6/65"         descriptive label
*   1                       file, observe control word
*   NAME1                   the file name
*   %"NAME2:9/6/65"         next label
*   2                       this file octal
*   NAME2                   second file name
*   %"BLOCK 322 ETC."       third label
*   2                       octal
*   322                     blocks 322
*   4                       through 325
*   %"DATA"                 last label
*   3                       unit 0, octal
*   0                       block 0
*   1                       only
*   ;
    <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 73_8, and the data block as a whole is terminated by 74_8.

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:

sense switch	meaning
0	read selected block
1	return to monitor
2	alphabetic format
3	index format
4	octal format

Potententiometer summary:

pot	meaning
0-2	block number
3	octal word size
4-6	line number
5-7	skip parameter

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 position 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 edge of the scope. 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:

sense switch	meaning
0	read selected record
1	return to monitor
2	density selection (on=hi)
3	format selection (on=binary)

Potentiometer summary:

pot	meaning
0-2	record selection number
3	octal word size
4-6	line number

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 Balgol-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:

e	v(e)
20	0020
JMP	6000
JMP+20	6020
JMP 20	6020
20JMP	6020
JMP20	??
JMP-20	5760
5-1+1	0005
5-1 1	0005
5-1-1	0003
STC	4000
ADD	2000
ADD STC	6000

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 μ is any tag and γ is a carriage return:

$e\gamma$ or $,\mu\ e\gamma$

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

comments

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

	symbolic line	absolute code
(a)	JMP 20 (START OVER)	6020
(b)	(THIS CAN MISLEAD)	0000
(c)	(THIS IS THE WAY TO HAVE VERY LONG, MULTI-LINE COMMENTS) JMP 30	6030

(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:

symbolic line	absolute code
"A "	4100
"A"	0041
"AA"	4141
"B"	0042
"B " - "A "	0100

*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 occurence of a ; in an expression has the same effect as the occurence 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=eγ

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

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

is a permissable 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 eγ

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 occurence 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:

location	symbolic line	absolute code
0520	JMP.1	6521
0521	JMP.-2	6517

GLOBAL

Texts are symbolically isolated from one another, i.e., tags occuring 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.

	location	symbolic line	absolute code
	--	GLOBAL A B	--
	--	ORG 500	--
text 1:	0500	,A JMP.1	6501
	0501	JMP A	6500
	0502	B+1	0507
	--	GLOBAL A	--
text 2:	0503	ADD A	2500
	0504	B	0505
	0505	,B	0000
	--	GLOBAL B	--
text 3:	0506	,B 4-3	0001
	0507	STC B	4506

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.

CONT

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

e_1
DITTO e_2

instructs Lass to assemble the expression e_1 and to place its value into $v(e_2)+1$ 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 ,μ7 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 1 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.

Text no.	Order of Assmbly	Location	Symbolic Line	Abs. Code
		--	ORG 1000	--
		1000	LDA	1000
	1st	1001	A	2000
		1002	STC 20	4020
1				
		--	VARB	--
		--	ORG 2000	--
		2000	,A 37	0037
	4th	--	DITTO 4	--
		2001	--	0037
		2002	--	0037
		2003	--	0037
		2004	--	0037
		--	GLOBAL A	--
	2nd	1003	STA	1040
		1004	B	2005
2				
		--	VARB	--
	5th	2005	,B	0000
		2006	,A 3	0003
		--	GLOBAL A	--
3	3rd	1005	LDA;	1020
		1006	A-3	2003

UPPER

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:

location	symbolic line	absolute code
--	ORG 400	--
0400	MSC 10	0010
0401	JMP A	6001
--	ORG 2001	--
--	UPPER	--
2001	,A JMP.1	6002
2002	MSC 10	0010
2003	JMP 20	6020
2004	A	0001

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:

```
0400    0063    (SET;3)
0401    0027
0402    6404    (JMP COMP3)
0403    6400    (JMP A)
0404    1000    (LDA)
0405    0003
0406    0017
0407    4003
0410    6000
```

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 occurence:

(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 over-sight 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.

ORG 2000

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 transfering 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 usefullness can be assembled and used without updating the program index.

Inherently Defined Lass Identifiers*

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

*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. I11, 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 i-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 AZEi$
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. I1I, ... , I7I 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(I21) = M(I1I)  $
COMMENT  LDAi1 , STAi2  $
```

Thus the value of the indicated index register is incremented <u>before</u> 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.)

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 InI=<variable> . The code generated is:

```
SET i n
<variable location>
```

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

```
STRING S(20)=( alpha string )$
I7I="3377" $
COMMENT: There is canonical correspondnece between
         registers and quarters $
I2I=S(0) $ I2=I2+"4000"  $
REPEAT 40$ M(I7I)=MH(I2I) $
COMMENT: LDHi2 , STAi7 $
```

D. POINTER is that cell ("155") in QUARTER 0 which points to the top of the BUFFER. The statement POINTER=M(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(POINTER)+"4001"$
```

would allow MH(121) to reference successive characters of that first item.

input-output

Since the Balgol 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>>
```

QUARTER 0 "/" indicates data location

```
0000 16
0001 CLR
0002 STA
0003 3140
0004 STA
0005 3000
0006 RCG
0007 4151
0010 JMP 400
0011 HLT
0012 HLT
0013 HLT
0014 HLT
0015 HLT
0016 HLT
0017 HLT
SAVE 0020 SET↑15
0021 3000
0022 LDA↑
0023 1776
0024 ADD 0
0025 STA↑15
0026 STC 16
0027 LDA↑16
0030 ROL↑1
0031 SCR 1
0032 STC 14
0033 LDA 14
0034 STA↑15
0035 LZE↑
0036 JMP 27
0037 LDA
0040 21
0041 STA↑15
0042 LDA
0043 15
0044 STC 21
0045 JMP 116
RESTORE 0046 LDA
0047 0
0050 STC 73
0051 SET 15
0052 21
0053 LDA 15
0054 AZE↑
EMPTY 0055 HLT
0056 STA
0057 21
0060 STC 15
0061 LDA↑15
0062 STC 14
0063 LDA↑14
0064 ROL↑1
0065 SCR 1
0066 STC 16
0067 LDA↑15
0070 STA 16
0071 LZE↑
0072 JMP 63
0073 JMP 0
PARAM 0074 ADD 23
0075 STC 17
0076 ADD 0
0077 ADD 23
0100 STC 16
0101 LDA↑16
0102 SCR 6
0103 STC 15
0104 LDA 16
0105 BCL↑
0106 7700
0107 ADD 117
0110 ADD 17
0111 JMP 113
0112 LDA↑17
0113 STA↑16
0114 XSK↑15
0115 JMP 112
0116 LDA↑
0117 6001
0120 ADD 16
0121 STC 122
0122 JMP 0
REPEAT 0123 LDA
0124 0
0125 STC 210
0126 JMP 20
/0127 SAM↑15
/0130 HLT 10
/0131 STC 136
0132 ADD 210
0133 JMP 74
/0134 JMP 1502
/0135 HLT
/0136 HLT
0137 LDA↑17
0140 AZE
0141 APO
0142 JMP 150
0143 COM
0144 STC 10
0145 JMP 135
STEP 0146 XSK↑10
0147 JMP 135
EXIT 0150 LDA
0151 136
0152 STC 243
0153 JMP 241
0154 SET↑11
POINTER 0155 3140
0156 LDA 11
0157 JMP 0
0160 LDA
0161 11
0162 STC 155
0163 JMP 0
PUT 0164 LDA
0165 0
0166 STC 176
0167 JMP 154
0170 LDA
0171 11
0172 STA↑11
0173 JMP 160
0174 COM
0175 STC 276
0176 JMP 0
0177 JMP 154
0200 STC 210
0201 ADD 17
0202 STA 11
0203 LDA↑13
0204 STA↑11
0205 XSK↑17
0206 JMP 203
0207 LDA↑
0210 0
0211 STA↑11
0212 JMP 160
0213 JMP 231
GET 0214 LDA
0215 0
0216 STC 176
0217 JMP 20
/0220 STC 301
0221 JMP 154
0222 AZE↑
EMPTY 0223 HLT
0224 STC 301
0225 JMP 175
LST.OP 0226 STC 243
0227 ADD 0
0230 STC 271
0231 JMP 20
/0232 ROL 3
/0233 STC 276
0234 JMP 271
GET.CL 0235 ADD 0
0236 STC 243
0237 JMP 154
0240 STC 155
LST.CL 0241 JMP 46
0242 CLR
0243 JMP 0 OVERLAY
STRING 0244 ADD 0
0245 JMP 74
/0246 JMP 1601
/0247 HLT
0250 LDA↑17
0251 STC 13
0252 ADD 247
0253 STC 271
0254 JMP 46
0255 LDA 13
0256 COM
0257 SRO
0260 276
0261 ADD 264
0262 JMP 274
LST.EL 0263 ADA↑
0264 7776
0265 STC 13
0266 ADD 0
0267 JMP 74
/0270 JMP 1601
/0271 HLT
/0272 JMP 46
0273 LDA↑17
0274 STC 17
0275 SRO↑
0276 0
0277 JMP 177
0300 SET↑11
0301 0
0302 LDA↑11
0303 BSE↑
0304 6000
0305 COM
0306 ADD 11
0307 STC 301
0310 LDA↑11
0311 STA↑13-
0312 XSK↑17
0313 JMP 310
0314 JMP 231
0315 HLT
0316 HLT
0317 HLT
0320 HLT
0321 HLT
0322 HLT
0323 HLT
0324 HLT
0325 HLT
0326 HLT
0327 HLT
0330 HLT
0331 HLT
0332 HLT
0333 HLT
0334 HLT
0335 HLT
RETURN 0336 JMP 46
0337 STC 341
0340 RCG
0341 0
0342 JMP 0
0343 AZE↑
0344 JMP 336
0345 APO
0346 JMP 363-
0347 STC 17
0350 ADD 0
0351 STC 342
0352 JMP 20
/0353 SCR 2
/0354 STC 356
0355 RCG↑
0356 4171
0357 ADD 130
0360 ADD 356
0361 STC 356
0362 LDA
0363 17
0364 NOP
0365 NOP
0366 NOP
0367 NOP
0370 NOP
0371 NOP
0372 ROL 3
0373 ADA↑
0374 4001
0375 STC 377-
0376 RCG
0377 0
```

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
0 (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 PARAMS 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. PARAMS 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. PARAMS 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:

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

which is equivalent to:

```
,P LDA
0
JMP PARAMS (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
```

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 executiøng 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
```

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 PUT(;;L1) , the BUFFER would have the following appearance:

LOCATION	CONTENTS	
0155	3154	
3140	0000	. . . null contents denote BUFFER bottom
3141	3776	
3142	0007	
3143	3776	
3144	0024	
3145	7771	
3146	4154	
3147	6050	
3150	4163	
3151	7474	
3152	0000	
3153	7474	
3154	3140	

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

```
(case 1) LDAi
         0005
         JMP OVERLAY
         JMP RETURN
```

is much more efficiently accomplished by:

(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:

	No. 1 Card:	SYSTEM	:	F-INFO
		TAPES	:	Mount on A3, at low density, a tape which can be removed from the Computation Center.
No.	No. 2 Card:	SYSTEM	:	F-INFO
	Control Card:	Cols. 1-6	:	BLINK3 file number (changes periodically), right justified.
		Cols. 7-11	:	BLINK

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 .

APPENDIX B

```
<PROG>>

?
.
.
.
.
.
.
.
TYPE
BLINK
DISTAPE
DISPLAY
DEFINE
MNTR
EDIT
.
.
.
.
QA
LASS
$
<PROG>>
```

```
LASS                                    <TXT 2
STACK #, FILES>>

10
/0
TYPROG
TYPROG2                                 <TXT 3
TYPE                                    1237 T1
GENTYPEN                                1132 E277
T1FAST                                  1134 RTRN
;                                       1127 NORMAL
<RUN>>                                  1120 NEWLINE

CORE USED:   400 1271


<TXT 1
1105 TYPE
1135 TYPEN
 200 X
6000 J
2000 A                                  <TXT 4
1000 L                                  1250 EM1
4000 S                                  1141 RET
 654 BL                                 1142 VAL
 410 NWTXT                              1234 SKIP
 630 BELLS                              1163 BLANK
 614 OCTADE                             1202 SPECIFIC
 443 FMT                                1166 ROT2
 645 READ                               1153 ROT1
 766 NC                                 1170 START
 474 TXT                                1216 TY1
 762 OCTAL
 752 UNT0
 460 BFMT
 756 OTXT
 550 NAME
 704 LINE
 532 NWP
 604 LNUM
 735 FINP
 603 NMR
 562 NMSG                               <TXT 5
 575 AST                                1245 LOOP
 627 LNR                                1271 FAST
 621 TYN
 660 BLR
 675 NBL
 732 LRT
 733 FINTXT
 730 TYCR
 751 FPR
 741 E1
1014 NXB
1077 BY1
1076 BY2
1075 BY3
```

```
LASS                                    <TXT 3
STACK #, FILES>>                        1634 CONVTBL

11
/2
BLINK
/0
INREC                                   <TXT 4
CONVTBL                                 1532 NXT
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
```

```
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 CHARS
 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 PARMSG


<TXT 2


<TXT 3
```

```
<TXT 4
1004 RET
1005 REC
1006 MINL
1007 MAXV
1010 PERR
1070 GOBKWD
1037 GOFWD
1107 FWD
1166 AREC
1143 APAR
1126 STOP
1054 RPAR
1052 CNCH
1152 LPAR
1114 BKWD
1120 BTH
1153 APRET
1154 APWT
1150 LPF
1247 ARET
1173 ARST
1252 HID
1177 LOD
1244 NOMO
1331 RWRT
1305 TOP
1317 TIM

<TXT 5
1336 NIR
1337 IM
1340 DUN
1433 IMAX
1374 NL
1403 NCH
1361 ADR
1364 E73
1400 NOC
1371 NEWL
1456 DUN1
1426 NRML
1432 ADD1
1462 E77
1455 E74
1463 NEG
1503 ABS
1474 ADNR
1475 NMBR
1510 NXT
1521 ST
1526 NT
1547 ADSR
1553 ADSN

<TXT 6

<TXT 7
1572 DISRET
1577 D1
1603 NEWLINE
1611 NEWCH
1636 BLANK
2122 CODES

<TXT 10
1647 RET
1650 VALSET
1651 NEW
1660 SCANPOTS
1711 TEST
1655 NEWIM
1717 STO
1664 SCRET
1701 SC2
1670 E100
1700 SAMN
2322 POT
1703 E1

<TXT 11
```

```
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
 404 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
 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
```

```
<TXT 2
1122 HEDR
```

```
<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 SCANPOTS
1525 TEST
1471 NEWIM
1533 STO
1500 SCRET
1515 SC2
1504 E100
1514 SAMN
2421 POT
1517 E1
```

```
LASS
STACK #, FILES>>

14
/2
DEFINE
;
<RUN>>

CORE USED:  400 1405



<TXT 1
```

```
LASS
STACK NO.,FILES>>

15
/0
VARBORG
/2
MONITOR
/0
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
1770 NOCR
```

```
LASS
STACK #, FILES>>

16
/0
EDIT1
EDIT2
EDIT3
EDIT4
T1
;
<RUN>>

CORE USED:    20 1724
```

```
<TXT 1
1700 T1
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
6000 GO
  20 READ
  32 TPIN
 404 BLK
  33 BN
 436 NBLKS
  55 AFTP
1213 SETOP
  56 WRITE
  61 MON
 472 BKMON
  65 CODES
 346 HED
 457 PARS
 434 WRT
1435 TMAX
 475 DEL
1122 VAL
1216 E1
 506 DN
 513 DTST
 526 DMV
 523 DINC
```

```
<TXT 2
 534 CLAL
1415 CHNGC
1020 INPTC
 541 USE4
 543 DISP
1153 FINDL
 605 E0
 612 NDG
 626 M1
 714 AC
 705 DISL
 712 EM40
 667 EM50
 760 TELE
 717 NCH
 751 DBLK
 763 PULSE
 765 WAIT
 777 IN
1011 NOCR
1375 IPCH
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 TYPT
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
```

```
LASS
STACK NO.,FILES>>

23
/0
VARBORG
/2
0A
/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
/0
VARBORG
/2
LASS1
;
<RUN>>

CORE USED:  400 2775



<TXT 1



<TXT 2
```

```
LASS
STACK #, FILES>>

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

CORE USED:  400 2200




<TXT 1




<TXT 2
```

```
<>

<PROG>>
```

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

```
LASS
STACK #, FILES>>

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

CORE USED:   400 1163
```

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

```
<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 4
1145 LOOP
```

THIS PROGRAM IS STORED ON
BLOCKS 230 AND 237 OF UNIT 0.

```
/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
T1FAST 1 30
TYPE 2 32
TYPROG 4 60
TYPROG2 1 65
VARBORG 1 73

PARAMS>>
```

SYMBOLIC ASSEMBLY PROGRAMS

```
/2
?

NAME/LENGTH/BLOCK NO.

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

PARAMS>>
```

BLINK FILES

```
**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 ADD1
LDA;
1
STC NCH
JMP NEWL
,NRML SET 15

<100>
17
JMP ADD1
JMP ADR
,ADD1 LDA;
,IMAX
ADD IM
AZE;
JMP 0

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

```
<120>
7400
JMP DUN
LDA
14
STC IM
JMP Ø
,E74 74
,DUN1 SET;15

<130>
E74
JMP ADD1
JMP DUN
,E77 77
,NEG JMP ADDC
"-"
LDA 11
COM

<140>
JMP ABS
,ADDN LDA
Ø
JMP PARAMS
7501
,ADNR
,NMBR
SET;11

<150>
NMBR
LDA 11
AZE;
CLR
,ABS STA 11
APO
JMP NEG
SET;12

<160>
-3
,NXT BCL;
777
AZE
JMP ST
LDA 11
ROL 3
STA 11

<170>
XSK;12
JMP NXT
,ST LDA
12
ADA;
7776
STC 12
,NT LDA 11

<200>
ROL 3
STA 11
BCL;
7770
ADA;
"Ø"
STC .2
JMP ADDC

<210>
(DIGIT)
XSK;12
JMP NT
JMP ADNR
,ADDS LDA
Ø
JMP PARAMS (%STR)
7601

<220>
,ADSR
LDA;17
STC 11
LDH;11
,ADSN LDH;11
SHD;
7400
JMP ADSR

<230>
SHD;
"%%"
JMP ADSR
STC.2
JMP ADDC
(CHAR)
JMP ADSN
```

```
**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>
ADA;
4000
STC D1+1
,D1 SET;12
(LOC OF STRING)
SET;13
1340
,NEWLINE SET;1

<20>
20
LDA;
-40
ADM
13
,NEWCH LDH;12
AZE;
JMP BLANK

<30>
SHD;
7400
JMP DISRET
SHD;
7300
JMP NEWLINE
ROL 1
ADA;

<40>
CODES
STC 14
LDA;
4
ADD 1
STC 1
ADD 13
DSC 14

<50>
DSC;14
JMP NEWCH
,BLANK LDA;
24
ADD 1
STC 1
JMP NEWCH
VARB

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

<70>
7731
2245
231
1304
1007
7
6000
60

<100>
3600
41
4100
36
3625
2517
3704
404

<110>
100
2
404
404
300
3
201
1004
```

```
<120>
4136
3641
2101
177
4526
3145
5141
2651

<130>
1070
1037
5171
4651
5136
1611
4241
7044

<140>
5126
2651
4530
3446
1200
0
100
12

<150>
1204
21
1212
1212
2100
412
2000
3045

<160>
2000
3056
4437
3744
5177
2651
4136
2241

<170>
4177
3641
5177
4141
5077
4050
4136
6745

<200>
1077
7710
7741
41
4142
4076
477
2112

<210>
177
101
2057
5720
1037
3704
4177
7741

<220>
4477
3044
4136
3743
4477
3146
4531
2245

<230>
7740
4040
177
7701
374
7402
275
7502

<240>
3663
6336
770
7007
4543
6151
0
0

<250>
301
1707
2000
3045
2000
3045
7777
7777

<260>
```

```
**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 BLKG
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

<100>
AZE;
CLR
S SKP
L;
,BLKG -1
AZE;
J.3
APO

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

<120>
7000
S TP 1
ROL;4 (GET UNIT)
ADA;
RDC
S TP
A BLKG
S BLKIN

<130>
,TP RDC
0
,NORD L;
6777
A SKP
ROR 1
S ST7
A SKP

<140>
AZE;
J BCD-2
J ADDS
SKPS
J ADDN
,SKP
SNS 2(BCD?)
J NO2 (NO)

<150>
,BCD 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
```

```
<170>
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>
LDH;7
J INT
,A4 J HED
LDH 7
S.2
J ADDC
(CHAR)
LDH;7

<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

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

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

XSK;1

<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 DISP1
,BIN L;
SNS 4
S SENSE
,B2 J TSTW
L
LINW
A M1

<330>
MUL;
,GRP
COM
A SKP
STA
ADT7
ADA;
-776

<340>
APO
J B3
L;
-1
ADM
LINW
J B2
,B3 J HED

<350>
L;
,ADT7
ROR 1
ADA;
7377
S 7
,NWL 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;
"0"
S.2
J ADDC
(DIG)
LDH 2

<400>
BCL;
7770
SRO;
2525
J CD
SRO;
,GRP1
J ADDB

<410>
XSK;1
J NHW
,CHARS SET 1
GRP
SET 2
7
J ADDC
,E73 73

<420>
L;
5252
S GRP2
,NC LDH;2
J TEST2
J DISP1
LDH 2
ADA;

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

<440>
J ADDC
(CHR)
,BI1 J ADDB
SRO;
,GRP2
J ADDB
XSK;1
J NC

<450>
J ADDC
73
SET 7
2
J NWL
,DISP1 J ADDC
74
,DISP J DISPLAY

<460>
IMAGE
SNS;1
J RETURN
J POTS
VAL-1
J NEW
SNS;0
J NEW

<470>
,SENSE
J NEW
J DISP
,VAL
DITTO 7
CONT
```

```
**DISPROG2**

,HED L
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
" "

<50>
J 11
,USEBLK J ADDC
77
J BI1
VARB
ORG 2000
7474
,IHED 10

<60>
"NA"
"ME"
"/L"
"EN"
"GT"
"H/"
"BL"
"K#"

<70>
7474
,SKPS 3
"SK"
"IP"
74
,IMAGE 200
DITTO 200
7474

<100>
```

```
**DISTAPE**

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

<10>
J.2 (NO RWD 1'ST 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 HEDR
,CHOOSE SNS 3(BCD,BIN?)
J BCD
LDA;
SNS 3

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

<50>
10
AZE;
ADD E1
COM
STC GRP
ADD 4POT
ROL 5
ADD 6POT

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

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

<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 USEBLK
ADD E73
AZE;
CLR
STC.2
J ADDC
(CHAR)
```

```
<170>
J ADDB
SRO;
,GRP2
J ADDB
XSK;4
J NC
LDH;2
J NXTL

<200>
,RDY1 J ADDC
74
,RDY J DISPLAY
IMAGE
SNS;1
J RET
SNS;4
J READ

<210>
,SENSE
J NEWIM
J POTS
POT-1
J NEWIM
SNS 0
J RDY
SNS;0

<220>
J.-1
J MOV
,RET RDC;
150
J 0
,TEST2 LDA
2
COM

<230>
ADD 7
ROL 1
AZE
APO;
XSK;0
J 0
,ADDB SET 11
0

<240>
J ADDC
" "
J 11
,USEBLK LDA;
77
XSK;0
J 0
,BCD LDA;

<250>
SNS;3
STC SENSE
ADD 4POT
ROL 2
ADD 6POT
ADD E1
,AL STA
LINE1

<260>
COM
STC 4
SET;2
LASTCELL-4000
,NL XSK;4
J.2
J BEGN
,SC LDH;2

<270>
J TEST2
J LOWLA
J CONV
SAE;
73
J SC
J NL
,BEGN J ADDN

<300>
,LINE1
J ADDC
73
,NEWC LDH;2
J TEST2
J RDY1
J CONV
AZE;

<310>
J BLNKS
,USC STC.2
J ADDC
(CHAR)
J NEWC
,BLNKS LDH;2
J TEST2
J RDY1

<320>
J CONV
AZE;
J BLNKS
J ADDB
J CONV
J USC
,LOWL LDA;
-1

<330>
ADD LINE
J BIL
,LOWLA LDA
4
ADD LINE1
J AL
,CONV LDH 2
ROR 1

<340>
ADA;
CONVTBL
STC 17
LDH 17
J 0
,HEDR LDA
0
STC HEDRT

<350>
J NEWIMAGE
IMAGE
J RDY
LDA
POT
ROL 4
ADD 2POT
ADA;
```

```
<360>
-40-1000
STA
REC
STC,2
J ADDN
(REC)
J ADDC
","

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

<400>
,POT
DITTO 7
VARB
,PARMSG 4
"PA"
"R "
"ER"
"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 LBUF 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
SET;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

<60>
STC AFTP
J WRITE
,CODES 0
0
7500
0
6000
6000

<70>
2476
6624
7731
2245
231
1304
1007
7

<100>
6000
60
3600
41
4100
36
3625
2517

<110>
3704
404
100
2
404
404
300
3

<120>
201
1004
4136
3641
2101
177
4526
3145

<130>
5141
2651
1070
1037
5171
4651
5136
1611

<140>
4241
7044
5126
2651
4530
3446
1200
0

<150>
100
12
1204
21
1212
1212
2100
412

<160>
2000
3045
2000
3056
4437
3744
5177
2651
```

```
<170>
4136
2241
4177
3641
5177
4141
5077
4050

<200>
4136
6745
1077
7710
7741
41
4142
4076

<210>
477
2112
177
101
2057
5720
1037
3704

<220>
4177
7741
4477
3044
4136
3743
4477
3146

<230>
5121
4651
7740
4040
177
7701
374
7402

<240>
275
7502
3663
6336
770
7007
4543
6151

<250>
0
0
301
1707
2000
3045
2000
3045

<260>
7777
7777
,LBUF
DITTO 47
,CAR SET 13
0
LDA;
345

<270>
J T1
LDA;
353
J T1
J 13
,HED"(  "
")  "
0

<300>
0073
ORG 400
J GET
J PARS
J GETCL
LDA;
,BLK
AZE

<310>
APO
J 336
ADA;
4000
STA
.3
STC WRT
RDC 10

<320>
0
SFT;2
2000
LDH;2
SHD;
"AA"
J WRT 1
LDA;

<330>
"AA"
STH 2
LDA;
74
STH;2
WRC 10
,WRT
LDA;

<340>
,NBLKS
ADA;
-4
APO;
CLR
ADA;
4
STA

<350>
NBLKS
ROL 10
ADA;
1777
COM
STC TMAX
RDC
160
```

```
<360>
J READ
,PARS ADD 0
J LSTOP
LDA;
BLK
J LSTEL
3776
LDA;

<370>
NBLKS
J LSTEL
3776
J LSTCL
,PKMON RDC
150
J
,DEL LDA

<400>
VAL
AZE;
ADD E1
COM
APO;
COM
STC 17
STC VAL

<410>
,DN SET 11
2
SET 3
2
LDH 11
,DTST SHD;
7400
J DMV

<420>
SHD;
7300
J DINC
LDH;11
J DTST
,DINC XSK;17
J.-3
LDH;11

<430>
,DMV STH 3
SHD;
7400
J DISAGN
LDH;3
J DMV-1
CONT
```

```
**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;
207
SRO
INPTC
J T1
SET;11
HED

<40>
LDA
3
BCL;
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
BCL;
7770
ADD E0
STH;11

<70>
XSK;17
J NDG
LDA;
,M1 -1
ADD 6
COM
ADD 4
COM

<100>
STC.3
STC VAL
,DISAGN SET;12
0(# OF LINES)
LDA;
1340
STC AC
SET;11

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

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

<130>
SRO
INPTC
ADD EM40
SET;17
,EM50 -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;
,EM40 -40

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

```
<170>
7300
J 15
SHD;
7400
J DBLK
ROL 1
ADA;
CODES

<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
,TELE SET;17
1241
CLR

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

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

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

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

<270>
,SUB1
J SB1 (SBST MODE)
J LGL (CNTRL MODE)
J CRBT
SHD;
7300
J LNUM
AZE;

<300>
J LNUM
SHD;
"AA"
J AH
SHD;
"BB"
J BK
SHD;

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

<320>
SHD;
"II"
J INPT
SHD;
"CC"
J CHNG
SHD;
"SS"

<330>
J SUBST
SHD;
"WW"
J WRITE
SHD;
"MM"
J MON
SHD;

<340>
"TT"
J TYP
SHD;
"LL"
J PLIN
SHD;
","
J CMA

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

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

<370>
J DISAGN
,LGL 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

<430>
SET;2
2001
SET;4
1
,TSK XSK;11
J.2
J
LDH 2

<440>
SHD;
7400
J 2TSK
SHD;
7300
J.3
LDH;2
J.-4

<450>
XSK;4
LDH;2
J TSK
CONT
```

```
**EDIT3**

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

<10>
J CLAL
,E1 1
,AH LDA
VAL
AZE
STC PG
LDA;
,PG 1

<20>
ADD 4
J DISP
,BK 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
2
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

<100>
LBUF
LDA;
73
STH;7
J DEL
,FIND CLR
COM
STC FINDC

<110>
J SUBST
,FNCH J FINDIT
J CLAL
,CMA CLR
COM
STC FINDC
XSK;4
ADD 4

<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 TYP5
SET;1
LBUF
,KPCK LDH;1
SHD 6

<150>
,FDTR (12=BEG,11=END)
SHD;11
J KPCK
J NWTRY
,FRBT J 0STS
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 RBOT
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
XSK;4
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
```

```
<310>
LDA;
147
J T1
LDA;
147
J T1
J CLAL
,RBOT LDA; (BKSPAC)

<320>
-4000
ADD 7
SAE;
LBUF
STC 7
LDA;
73
STH 7

<330>
J DISAGN
,TYP5 SRO
FINDC
J CLAL
SET 4
5
LDA;
,PLIN J CAR

<340>
,TYP4 SET;1
-4
LDA
4
ROL 3
STA
4
BCL;

<350>
7770
ROL 1
COM
ADA;
237
J T1
XSK;1
J TYP4 2

<360>
J CAR
SRO
SUB1
J SETOP
J CLAL
```

```
**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 OCCURNCE)
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

<10>
STC SKIP
STC BLANK
LDA;17
AZE
JMP SPECIFIC
SET;11 (USE MINIMUM FIELD)
-4
JMP ROT2

<20>
,ROT1 LDA
VAL
BCL;
0777
AZE
JMP START
ADD E277
SRO;

<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)

<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
```

```
**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 BSE;
200
,WAIT XSK;17
JMP.-1
LZE
JMP IN

<20>
ROR;1
SET;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<N$MINL,MAXV$ERR>
   N : NO. REC'S TO MOV
   MINL: 1'ST LOC TO FIL
   MAXV: SET TO MAX LOC

<10>
   ERR: ERROR EXIT

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

<20>
,RET
,REC
,MINL
,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>
,BKWD 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
,AREC 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
```

```
<240>
SXL;4
JMP LOD
SXL;4
JMP LOD
SXL;4
JMP LOD
SXL;4
JMP LOD

<250>
SXL;4
JMP LOD
SXL;4
JMP LOD
SXL;4
JMP LOD
SXL;4
JMP LOD

<260>
,NOMO LDA
11
STA 15
,ARET(RET)
SXL 4
JMP.-1
,HID OPR 4
XSK 11

<270>
STH;11
SET;17
7762
SXL;4
JMP HID
XSK;17
JMP.-3
JMP NOMO

<300>
,REWIND LDA
0
STC RWRT
ATR
SET;17
7543 (5 MS. DELAY)
XSK;17
JMP.-1

<310>
LDA;
60
ATR
XSK;17
JMP.-1
CLR
ATR
SXL;5

<320>
JMP.-1
,TOP MUL 1 (WAIT 2 SEC)
SRO;
3777
JMP TOP
XSK;17
JMP TOP
SXL;5

<330>
JMP TOP (FIND LOAD PT)
JMP STOP
JMP FWD
,TIM SET;17 (MOV 3")
-405 (=D 261)
XSK;17
JMP.-1
SNS 2

<340>
JMP RWRT-1
SRO;
3777
JMP TIM
JMP STOP
,RWRT
```

```
**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 L0
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

J TYPE
"U"

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

<50>
J TYPE

J TYPEN
,L0
4
J TYPE

J TYPEN

<60>
,LN
4
J TYPE
73
SNS 5
J SPC
J.4
,RET RDC;

<70>
150
J
SET;5
6777
RF(BSAVE)
,NXTB SNS 5
J SPC
R

<100>
ROL 6
APO
SKP-1
ROR 6
SHD;
1000
SKP-1
SHD;

<110>
1100
SKP-1
SHD;
1300
SKP-2
SHD;
1200
SKP-2

<120>
SHD;
1400
J NTXT
SHD;
200
J TAG
SHD;
400

<130>
J SPC
SHD;
1500
J BLINK
SHD;
100
J CR
J NXTB

<140>
,NTXT J TYPE
73
J SPAC
J TYPE
"<"
J TYPE
"T"
J TYPE

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

J TYPEN
1
0

<160>
LDA;
1
ADM
.-5
J TYPE
73
 J NXTB
,TAG R
```

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

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

<210>
J TYPEN
(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
RCG;
SET;2
6221
SET;3

<240>
WCG;
LDA
L0
SCR 10
STA
4
COM
STC 0

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

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

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

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

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

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

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

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

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

```
<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
BCL;
7770
ROL 1
COM
ADA;

<30>
237
SRO;
3567
JMP 0
JMP T1
,RET JMP
```

```
**POTS**

GLOBAL POTS
,POTS LDA
0
JMP PARAMS (SVALS-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
LDA;17
STC SC2
LDA;
,E100 SAM 0

<30>
STC SAMN
SET;11
POT-1
SET;12
7767
SET 13
VALSET
,SAMN

<40>
,SC2(JMP TEST OR STO)
LDA;
,E1 1
ADM
SAMN
XSK;12
JMP SAMN
JMP SCRET

<50>
,TEST COM
ADA;11
SCR 1
AZE
JMP NEWIM
JMP 0
,STO STA;11
ADA;

<60>
200
SCR 2
STA;13
JMP 0
VARB
,POT 0
DITTO 7
```

```
**PROGSTACK**

.$.$
.$.$
.$.$
.$.$
.$.$
.$.$
.$.$

<10>
TYPE$(MSG,FMT,FILE)'S$
BLINK$FILE$
DISTAPE$NIL$
DISPLAY$NIL$
DEFINE$1 STRING$
MNTR$.$
EDIT$TEXT$
.$.$

<20>
.$.$
.$.$
.$.$
QA$.$
LASS$STACK #, FILES$
$$
```

```
**T1**

GLOBAL T1 EM1
,T1 SET 15
0
SNS;0
JMP.-1
SET;16
-13

<10>
,LOOP STA;
0
BCL;
,EM1 -1
ATR
SET;17
7355
XSK;17

<20>
JMP.-1
LDA
LOOP+1
SCR 1
XSK;16
JMP LOOP
JMP 15
```

```
**T1FAST**

GLOBAL T1 EM1
,T1 SET 15
Ø
SNS;Ø
JMP.-1
SET;16
-13

<10>
,LOOP STA;
Ø
BCL;
,EM1 -1
SNS;2
ROL 1
ATR
LDA;

<20>
7356
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
XSK;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
4
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

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

<110>
"LG"
"L "
"FM"
"T "
7474
,TXT J READ
SET;7
3400

<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
```

```
<170>
J NWP
,NAME L
0
S NMR
SET 2
1
LDH;2
J TYPE

<200>
"*"
J TYPE
"*"
,NMSG LDH;2
SHD;
7400
J AST
SHD;

<210>
7300
X;5
S.2
J TYPE
(CHAR)
J NMSG
,AST J TYPE
"*"

<220>
J TYPE
"*"
J BL
1
,NMR
,LNUM L
0
S LNR

<230>
J BL
1
L;
10
ADM;
,OCTADE
S TYN
J TYPE

<240>
"<"
J TYPEN
,TYN
0
J TYPE
">"
J BL
1

<250>
,LNR
,BELLS J TYPE
-31
J TYPE
-31
J TYPE
-31
J TYPE

<260>
-31
J TYPE
-31
J TYPE
-31
J RETURN
,READ RDC 10
(BLK)

<270>
L;
1
ADM
READ 1
J 0
,BL L
0
J PARAMS

<300>
7601
,BLR
SNS;1
J BELLS
L;17
COM
S 2
J TYPE

<310>
73
X;5
NOP
X;2
J.2
J BLR
,NBL J TYPE
-26

<320>
X;5
NOP
X;2
J NBL
J BLR
,LINE L
0
S LRT

<330>
A 7
AZE
J.4
J READ
SET;7
7377
LDH;7
SHD;

<340>
7400
J FINTXT
SHD;
7300
J TYCR
S.2
J TYPE
(CHAR)

<350>
J 3LINE
,TYCR J BL
1
,LRT
,FINTXT J FINP
J NWTXT
,FINP L
0
```

```
<360>
S FPR
J BL
,E1 1
J TYPE
-26
X;5
J.-3
J NAME

<370>
J BL
14
,FPR
,UNT0 L;
RDC
S READ
J OCTAL
,OTXT L;

<400>
-1
A 1READ
S 1READ
,OCTAL SET;5
-67
J NAME
SET;3
,NC

<410>
J TYPE
"<"
J TYPE
"U"
J TYPE
"N"
J TYPE
"I"

<420>
J TYPE
"T"
J TYPE
" "
L
READ
SCR 3
BCL;

<430>
-1
S.2
J TYPEN
0
0
,NXR J READ
J BL
1

<440>
J TYPE
"<"
J TYPE
"B"
J TYPE
"L"
J TYPE
"K"

<450>
J TYPE
" "
L
1READ
ADA;
-7001
S.2
J TYPEN

<460>
0
0
SET;7
3377
J REPEAT
J BY1
4
J BL

<470>
1
J REPEAT
J BY2
10
J BL
1
J REPEAT
J BY3

<500>
10
L;7
S.2
J TYPEN
0
4
CONT
```

```
**TYPROG2**

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

<1Ø>
,BY3 J STEP
,BY2 J STEP
,BY1 J FINP
SET;5
-67
X;3
J NXB
J NWTXT

<2Ø>
```

```
**VARBORG**

VARB
ORG 2000
```

APPENDIX C

```
                STANFORD BLINK COMPILER - - VERSION OF 20 MAY 65
      COMMENT  THIS IS PASS 1 OF THE ASSEMBLERS
     *-SPACE
   ...    GLOBAL NEXTEXT, GSW$
   ...    GLOBAL TXTN$
   ...   GLOBAL CON,MODE,BLK,BASE,CHAR  ,CONMAX,
   ...       T,SEARCH,VALGLB,GLBSW,BLINKSW,LINE,ESAVE,BADBLK$
   ...     GLOBAL  PSEUD , BUFFER  $
   ...    GLOBAL SRCHPARS()$
   ...     ARRAY BUFFER (10) $
   ...    ARRAY VALGLB(28)$
   ...    ARRAY BLK(0..3)=(''6200'',''6200'',0,''7200'')$
   ...    ARRAY CHAR(10)$
   ...
   ...        COMMENT
   ...     PARAMETERS ARE . . .  STK NO., TEXTS            $
   ...
   ...    COMMENT CORE USE PLAN IS . .
   ...            SYMBOL TABLE ARRAY ETC THRU QTR 5.
   ...            QTR 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 ..   BLKS= (200)+EXT BLK NO.
   ...          $
 11... PROCEDURE TAPE ( INST , INC , N ) $
 11... BEGIN
 11...    M(9)=INST  $
 13...    M(10)= INCR(INC,BLK(N)) $
 23...    M(11)=''6000'' $
 26...    DO ''6011'' $
 27... END TAPE() $
 27...
 30... PROCEDURE WRITE(INFO) $
 37... BEGIN
 37...    IF I6 EQL ''7377'' $
 42...        BEGIN   I6=''6777'' $
 46...            TAPE(''704'',1,1 ) $
 52...    END $
 52...    MH(I6I)=INFO $
 55... END WRITE() $
 55...
 55...
 56...  PROCEDURE WRITF(INFO)$
 65...     BEGIN  WRITE(SCLR(6,INFO))$ WRITE(INFO)$
 77...   END WRITF()$
 77...
 77...
100...     GLOBAL NEQL$
100...   PROCEDURE SEARCH($$FOUND)$
107...    BEGIN
107...    CON = M(I5I)+M(I5)$
113...       UNTIL M(I2I) EQL 0 $
```

```
114...        BEGIN
116...             I1I=CHAR(1)$
120...     I2N = I2 + (MH(I2) - '0')/2$
127...             IF M(I1) NEQ M(I2) $ GO NEQL$
134...     UNTIL MH (I1I) EQL 0$
135...     IF MH(I1) NEQ MH(I2I)$ GO NEQL$
150...     GO FOUND$
151...         NEQL..  I2=I2N$
153...        CON=CON+M(I5)$
153...
156...        END $
156...
157...        COMMENT NOT FOUND VALIDLY IN TABLE $
157...   END SEARCH()$
157...
163... PROCEDURE LOOKUP($$CONTROL,RESERVED,TAG)$
172... BEGIN
172...    COMMENT  CHAR IS SOUGHT  FIRST AMONG LOCALLY VALID GLOBALS.
172...      (IF FOUND THERE, SEE 'FOUND')
172...    IF ABSENT AND GLBSW ON,SYMBOL IS ADDED TO GLOBAL TABLE.
172...    OTHERWISE SOUGHT IN LOCALS. IN LATTER CASE, ORPHANS ALWAYS
172...    HOUSED.  IN ANY CASE SYMBOL FINALLY 'FOUND'..
172...          THE VALUE OF A SYMBOL (CON) MEANS . . .
172...               PSEUDO OP . .  SWITCH VALUE
172...               RESERVED WORD . . 6-BIT POINTER
172...               IDENTIFIER (TAG) . .  FULL-WORD POINTER
172...       $
172...
172...
172...    ARRAY T(''650'') = (
172... '6GLOBAL',
172...    '5UPPER',
172...     '6DEFINE',
172... '3ORG',
172...   '5DITTO',
172... '4VARB',
172...    '4CONT',
172...          0,
172...       '3HLT',
172...       '3CLR',
172...       '3MSC',
172...       '3ATR',
172...       '3RTA',
172...       '3NOP',
172...       '3COM',
172...       '3ROL',
172...       '3ROR',
172...       '3SCR',
172...       '3ADD',
172...       '3STC',
172...       '3JMP',
172...       '3LDA',
172...       '3STA',
172...       '3ADA',
172...       '3ADM',
172...       '3LAM',
172...       '3MUL',
172...       '3SAE',
172...       '3SRO',
172...       '3BCL',
```

```
172...        '3BCD',
172...        '3BSE',
172...        '3DSC',
172...        '3LDH',
172...        '3STH',
172...        '3SHD',
172...        '3SET',
172...        '3SAM',
172...        '3DIS',
172...        '3XSK',
172...        '3OPR',
172...        '3SNS',
172...        '3AZE',
172...        '3APO',
172...        '3LZE',
172...        '3IBZ',
172...        '3SXL',
172...        '3KST',
172...        '3RDC',
172...        '3RCG',
172...        '3RDE',
172...        '3MTB',
172...        '3WRC',
172...        '3WCG',
172...        '3WRI',
172...        '3CHK',
172...        '4SAVE',
172...        '7RESTORE',
172...        '6PARAMS',
172...        '6REPEAT',
172...        '4STEP',
172...     '7OVERLAY',
172...        '7POINTER',
172...        '3PUT',
172...        '3GET',
172...        '5GETCL',
172...        '5LSTOP',
172...        '5LSTCL',
172...     '6STRING',
172...        '5LSTEL',
172...        '6RETURN',
172...           0)$
172...
172...
172...        ARRAY SRCHPARS(7)=(0,1,256,-1,0,1,0)$
172...
172...
172...
172...        COMMENT FIRST SEARCH PSEUDO-OPS..        $
172...        I5I=SRCHPARS(1)$
174...      I2I=T(0)$
176...      SEARCH($$GOC)$
200...
200...        COMMENT NEXT SEARCH RESERVED WORDS..        $
200...     I5I = SRCHPARS(1)$
202...      SEARCH($$RESERVED)$
204...
204...     COMMENT NEXT SEARCH GLOBALS..  $
204...
204...        I5I=SRCHPARS(3)$
```

```
206...:        IF GLBSW$ GO HOUSE$
212...         SEARCH($$TSTGLB)$
214...         GO LOCL$
214...
214...
215...       TSTGLB..
215...            COMMENT  SEE IF AN ACTIVIZED  GLOBAL $
215...            I1I=VALGLB(0) $
217...            UNTIL M(I1I) EQL 0 $  IF M(I1) EQL CON $ GO GOT$
231...            COMMENT NO SUCH LUCK $
231...           GO NEQL $
231...
232...         LOCL..
232...      COMMENT FINALLY LOCALS..$
232...         I5I=SRCHPARS(5)$
234...         SEARCH($$GOT)$
236...         M(I5I)=CON$
241...         GO ESYM$
241...
242...       HOUSE..
244...       SEARCH($$GOT)$
244...       ESYM..
246...       WRITE(2)$
246...       I1I=CHAR(1)$
250...       WRITE(MH(I1)-'0')$
255...          MH(I2)=MH(I1)  $
257...          REPEAT (MH(I1) - '0' ) $ ( MH(I2I)=MH(I1I)$  WRITE(MH(I1))) $
273...
274...            M(I2I)=M(I2I)=M(I2I)=0$
300...            GOT..     GO TAG$
301...         GOC.. IF CON EQL 7$  GO NEXTEXT$
304...
304...
306...            WRITE(8)$
310...              PSEUD=1$  GO CONTROL $
314...
314...
315... END LOOKUP() $
315...
325...        PROCEDURE RETWR(STR())$
335...      BEGIN TAPE(''700'',0,0)$
341...       LIST BBL(STR(),TXTN  ,LINE)$
361...       M(''21'')=BSAVE$
364...       PUT($$BBL)$
366...       OVERLAY 0$
370...     END RETWR()$
370...
371...     STRING ILTXT(5)=('ILLGL TXT'),
371...         BDCHR(6) = ('BAD CHAR AT')$
371...
371...     GLOBAL ILTXT,BDCHR,PRSOF $
371...
371... PROCEDURE SCAN($$CONTROL,RESERVED,TAG,CONST)$
400...
400... BEGIN    COMMENT  I7 SCANS TEXT IN QUARTER  7  $
400...
422...      FUNCTION BTWN(X,Y)=(X LEQ MH(I7)) AND (MH(I7) LEQ Y)  $
422...
432...      SUBROUTINE STEP$
436...      BEGIN
```

```
436...        IF I7 EQL 0$
437...        BEGIN
437...           COMMENT END OF BLOCK$
445...           TAPE(''710'',1,2)$
445...           I7 = ''7377''$
447...        END$
447...        I7 = I7 + ''4000''$
453...     RETURN END STEP$
454...
454...     ARRAY CONS(10)=(''73,73,74,74'',
454...    ' .' , ' ,' , ' -' , ' =' , ' ' , ' +' )$
454...
454...     TOP..
472...         FOR CON =(1,1,10)$
472...           IF MH(I7) EQL CONS(CON)$
507...              BEGIN   ENTER STEP$  IF CON GTR 7$ GO TOP$
521...              GOC.. GO CONTROL        END$
523...        EITHER IF MH(I7) EQL ' ('$
526...           UNTIL MH(I7) EQL ')'$   ENTER STEP$
537...         OR IF MH(I7) EQL ''33''$
543...             BEGIN CON=16$ GO EC END $
551...     OR IF MH(I7) EQL 2$  COMMENT BCD CONSTANT$
555...           BEGIN
555...           CON = 0$
561...     NBCD.. ENTER STEP$
562...           IF MH(I7) EQL 2$
564...           GO EC$
566...           CON = 64CON + MH(I7)$
576...           GO NBCD$
577...     EC..  ENTER STEP$  GS..
600...                GOCON..                          GO CONST   $
601...           END$
601...        OR IF BTWN('0','7')$
605...           BEGIN
605...              CON=0 $
611...                  UNTIL NOT BTWN('0','7')$
614...                     (CON=8CON+ABS(MH(I7)-'0') $ ENTER STEP ) $
632...              GO GS$
633...           END$
633...        OR IF BTWN('A','Z')$
637...           BEGIN
641...                  I1I=CHAR(1) $
643...             I2=I1 $
645...              UNTIL NOT (BTWN('A','Z') OR BTWN('0','9')) $
655...                 BEGIN  MH(I1I)=MH(I7)$  ENTER STEP END $
663...              MH(I2) = 2(I1-I2)+'0' $ MH(I1I)= 0 $
674...              LOOKUP($$GOC,RESERVED,TAG)$
700...           END $
700...         OTHERWISE $ RETWR(BDCHR())$
704...        ENTER STEP$
704...
705...     GO TOP $
706... END SCAN()$
706...
706...
720... COMMENT PROGRAM BEGINS HERE..
720...        $
720...
720...
720...     I6=''6777'' $
```

```
 722...    TAPE(''704'',0,0)$
 726...        COMMENT QTR 6 NOW SAVED $
 733...        WRITF(M(''21''))$  I1I=BUFFER(0)$
 735...       BSAVE = M(''21'')$
 740...        M(''21'')=I1$
 742...       I3=M(POINTER) $
 746...          POINTER = I3$
 750...       DMY = M(I3I) $ COMMENT  SKIP CONTROL WORD IN BUFFER  $
 755...       WRITE(11) $
 755...       WRITF ( M( I3I ) ) $
 761...
 761...        I3=I3+256$
 765...    GO CLVGS$
 765...
 765...
 765...
 765...
 766...    NEXTEXT..TAPE(''700'',0,3)$
 772...       TXTN=TXTN+1$
1000...       DO(''223'')$    COMMENT SKIP CONTROL WORD IN BUFFER$
1001...    BLK(2)=BTSET(''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 BUFFERS$
1022...       DO(''223'')$    COMMENT IGNORE LENGTH OF TEXT$
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(13)  GO NEXTSYM END$
1056...    OTHERWISE $ RETWR(ILTXT())$
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
1101...            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()=64PT1+MH(I7)$
1142...        END FW()$
1142...
1143...            I7=I7-''4000''$
1147...            REPEAT FW(0) $
1155...            BEGIN  ENTER BW $ ENTER BW$ ENTER BW$  END $
```

```
1161...            FW(0) $  COMMENT 2+3FW(0)  OVERFLOWS LINC WORD  $
1165...            VRBWS.. IF FW(0) EQL 0$ GO NEXTEXT$
1167...            FW(0)$
1171...            GO VRBWS$
1172...         END$
1172...
1172...     NEXTSYM..
1172...         PSEUD=0$
1174...        SCAN($$CONTROL,RESERVED,TAG,CONST)$
1201...     RESERVED..
1203...         WRITE(9)$ WRITE(CON)$ GO NEXTSYM$
1211...     TAG.. WD=BTSET(''4000'',    CON)$
1217...        IF GLBSW$ (M(I4) =CON$ M(I4I) =0)$
1230...         WTAG.. WRITF(WD)$
1235...        GO NEXTSYM$
1236...         CONST.. WRITE(10)$  WD=CON$ GO WTAG$
1244...     CONTROL..
1251...        WRITE(CON)$
1251...             EITHER IF PSEUD$ CON = CON +4$
1261...             OTHERWISE$ CON=MIN(CON,4)$
1273...     SWITCH CON, (NEXTLINE,EOT,EOT,NEXTSYM,GLB)$
1316...           COMMENT CONTROL OF MEANING ONLY TO SECOND PASS $
1316...             GO NEXTSYM  $
1316...
1317...     EOT.. COMMENT END OF A TEXT $
1317...        COMMENT ERASE LOCAL SYMBOLS $
1317...
1321...           I2I=T(0)$     CHAR(1) =0$
1324...    EOT1..  SEARCH($$EOT1)$  COMMENT CONTROLS$
1330...    EOT2..  SEARCH($$EOT2)$  COMMENT RESERVED WORDS $
1332...    EOT3..  SEARCH($$EOT3)$  COMMENT GLOBALS  $
1332...           M(I2I)=M(I2I)=0$
1335...     SRCHPARS(5) = SRCHPARS(7)$
1341...     CLVGS.. I4I = VALGLB(0)$
1343...        M (I4I) = 0$
1343...
1345...         GO NEXTEXT $
1345...
1346...     GLB..  GLBSW=1$  GO NEXTSYM$
1352...     PASS2..
1354...     I1 = ''3400'' - 1$  REPEAT 256$  M(I1I) = 0$
1362...         WRITE(4)$
1364...        TAPE(''704'',1,1)$
1370...     OVERLAY -''25''$
1373...           COMMENT  ON TO  SECOND PASS  $
1373...
1373...
1373...
1373...  FINISH$
```

```
...
   *-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) .                  13(0) BLINK
...    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 .
...            'DEFINE' . . .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 1'ST 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 SCMO, SKPEQL, ONAM , LINE, CON, BADOP, BADOP1, UNDEF,LSTWRD $
...     GLOBAL  BLK, PASS, VARB, VRBSW, VABS, ABSL, LOCN, ERRSW,
...        NOWRITE, WORD, UNDEF, EOT , EOO, 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))  $
23...      M(11)= ''6000'' $    DO ''6011'' $
27...   END TAPE() $
27...
30...   COMMENT INITIALIZE INPUT TAPE $
30...      I5= ''6777'' $
30...
32... PROCEDURE READ(DMY) $
41... BEGIN
41...    IF I5 EQL ''6777'' $
43...    BEGIN    TAPE(''700'' ,1, 5 ) $  I5=''6377''  END $
53...    READ()= MH(I5I) $
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($$L)$(IF PASS NEQ 2 $ GO L )$
112...
116... PROCEDURE WRITE(INFO) $
125...   BEGIN
125...      INCR(1,M(I3))$
127...     IF PASS - VRBSW NEQ 3  $ RETURN $
140...     IF I7 EQL ''3777''$
143...     BEGIN TAPE(''704'',1,7)$ I7=''3377'' END $
153...     LSTWRD=M(I7I)=INFO$
160... END WRITE() $
160...
160...
161... PROCEDURE SCAN( $$ TAG , CONST , RESRVD , PSEUDO ) $
170... BEGIN
170...
170...    ARRAY VALUE (   63) =   (  ''
170...             0, 11, 0, 14, 15, 16, 17, 240,
170...       300, 340, 2000, 4000, 6000, 1000, 1040,
170...       1100, 1140, 1200, 1240, 1440,
170...       1500, 1540, 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...
```

```
176...     TOP..     IF READ(0) LSS 32 $  SWITCH MH(I5) ,
206...        (EOL, SKP, SCM0, EOO, DOT, COMMA, PSEUM,PSEU1,
206...        RSRVD, CONSTNT, PROGN,INTXT,BLINK
237...        )$
237...     COMMENT A TAG  $
246...        CON= SCLR(1, ROTR (5, MH(15))) +READ(0) $
251...        I1 =  CON + ''3000'' $  I2=I1 + 256 $
261...        IF I2 EQL UNDTAG$ GO UNTGMSG$
267...        WORD = WORD + SGN.M(I1)$
275...        IFN2( $$ GOT) $
277...        M(I2)= MAX ( M(I2) , 1) $
310...     GOT.. GO TAG  $
311...     DOT.. UNDEF = 1 - ABSL $
316...                WORD=WORD+SGN(M(I3)-UPPER) $
326...     GOR.. GO  RESRVD $
327...     COMMA..  CON=7$
332...     GOP.. GO PSEUDO $
333...     PSEUM..  SGN = -SGN $   GO TOP $
340...     RSRVD..  WORD=WORD+ SGN.VALUE(READ(0)) $ GO GOR $
354...     CONSTNT..   CON= SGN.READF(0) $     WORD=WORD+ CON $
364...        GO CONST $
364...
365...      SKP.. REPEAT(READ(0))$ READ(0)$ GO TOP$
377...      PROGN..  PROGNUM=READF(0)$
403...        GO TOP $
404...     INTXT.. BLOCK = BLOCK + 1$
411...     LINE = 1$
414...     GO TO TOP$
415...     PSEU1..  CON = READ(0)$    GO GOP $
421...     EOL.. LINE= LINE + 1 $
425... END SCAN() $
425...
437...      STRING DUPL(7)=('DUPL. TAG AT'),
437...        UNDF(8) = ('UNDEF. TAG AT'),
437...      ILLGL(6)=('ILLGL OP AT')$
437...      GLOBAL GOODAS,DUPL,UNDF,ILLGL,STK  $
437... PROCEDURE RETWR(STR())$
447... BEGIN    LIST L(STR(),BLOCK,LINE)$
467...        TAPE(''700'',0,6)$
473...     GLOBAL BSAVE $
473...     M(''21'') = BSAVE $
476...     PUT($$L) $
500...     OVERLAY 0 $
502... END RETWR() $
502...
502...
502...
502...
503... PROCEDURE DEFEXPR(DMY) $
512... BEGIN    WORD= 0 $
513...     NXT..    SCAN ($$ TAG, NXT, RSRVD, BADOP) $
520...        DEFEXPR() = WORD $   RETURN $
523...     TAG.. IFN2( $$ NXT) $
525...        IF M(I2) NEQ 2 $  GO BADOP $  GO NXT $
533...     RSRVD..  IF UNDEF $  GO  BADOP$  GO NXT $
540... END DEFEXPR() $
540...
541... SUBROUTINE GETAG $
545... BEGIN
552...     SCAN ( $$ TAG, BADOP, BADOP,BADOP ) $
```

```
552...           GO BADOP $
553...     TAG.. IFN2( $$ RET ) $
555...          IF M(I2) NEQ 1$ RETWR(DUPL())$
565...        M(I2) = 3 - ABSL $
573...   RET.. I4=I1$
575...    RETURN $
576... END GETAG  $
576...
576...
576...     ARRAY BLNKS(20) $
576...
576...
576... COMMENT  GET READY FOR PASS 2 $
576...
576...      PASS=2$
576...
601...     COMMENT  DEFINE FLAGS..
601...        1.. UNDEFINED
601...        2.. DEFINED
601...        3.. ADD VARB
601...                    $
601...
601...     GLOBAL ARRAY LOCN(0..1) =(''400''    )$      LOCNO = 256 $
604...     COMMENT LOCN(0) INDEXES CODE SPACE, LOCN(1) VARB SPACE.
604...        VABS DENOTES ENCOUNTERING OF 'ORG' IN VARB SPACE $
604...
607...    SCM2..BSAVE=READF(0)$ I1I=SAVBUF(1)$ M(''21'')=I1$
613...           BLOCK = 0$
613...
613...
613...
614... SCM0..    VRBSW= 0 $      I3I = LOCN(0) $  ABSL = 1 $
622...                 UPPER = 0$
623... SC0..      NOPS =           WORD =  0 $
627... SC..       SCAN ( $$ SC , SC , SC, PSEUDO ) $
634...        WRITE ( WORD) $   GO SC0 $
642...     UNTGMSC.. RETWR(UNDF())$
645... PSEUDO..     SWITCH CON,
672...     (GLB,UPR,DEF,ORG,DIT,VRB,COMMA)$
672...   GLB..
677...      SCAN($$ GLB,BADOP,BADOP,BADOP)$ GO SC0$
677...
700...     UPR..  UPPER=''2000''$
703...    BAS.. COMMENT EXPECT CARRIAGE RETURN ONLY  $
710...      SCAN($$BADOP,BADOP,BADOP,BADOP)$ GO SC0$
711...    DEF..
711...                          ENTER GETAG $
712...    IFN2($$SKPSET)$
714...        M(I2)= 2 $
717...    SKPSET..
722...       M(I4)= ( DEFEXPR( 0)) $      GO SC0 $
723... BADOP..   RETWR(ILLGL())$
726... ORG.. N= DEFEXPR(0) $
732...        IF NOT ABSL $  (  ABSL=VABS=1 $  M(I3)=N $ GO SC0) $
746...             IF (NOT VRBSW) AND (M(I3) EQL LOCNO) $
756...           BEGIN COMMENT THIS IS AN INITIAL ORG, IT SUPERCEDES
756...              ORG 400 ASSUMPTION AND MAY HAVE ANY VALUE $
756...              LOCNO = N $
763...              M(I3) =  BTCLR( ''377'',N) $
767...           END $
```

```
 767...            IF N LSS M(I3) $  GO BADOP$
 776...            LSTWRD = 0 $   N=N - M(I3) $  GO RPT1 $
1005... DIT..      N = DEFEXPR(0) $
1010... RPT1..     REPEAT N $  WRITE (LSTWRD) $  GO  SCO $
1025... VRB..      I3I = LOCN(1) $   VRBSW = 1 $
1032...              UPPER=0$
1033...            ABSL = VABS $  GO BAS $
1036... COMMA..
1037...          ENTER GETAG $  M(I4)=(M(I3)-UPPER)$
1044...          GO SCO$
1044...
1045... E00..      COMMENT A PASS NOW COMPLETED  $
1045...         VARB = LOCN(0)$
1051...         I3=LOCN(1)-1$
1056...            LOCN(0)=LOCNO $   LOCN(1)=VARB $
1065...            VABS=1 $
1070...         IFN2 ( $$ E001 ) $   COMMENT CK SYMBOL DEFS AFTER PASS 2 ONLY $
1075...         INCR(SCLR(8,LOCNO),BLK(7))$
1077...             I7=''3377''$
1101...      I1 = ''3400''$
1103...      E002.. IF I1 EQL ''3777''$ GO EP3$
1110...       SWITCH M(I1I) , (E003 , E002 , E004 ) $
1130...      GO E002$
1131...      E003..
1131...         UNDTAG = I1$
1134...         GO EP3$
1135... E004..     INCR( VARB , M(I1-256)) $  GO E002 $
1145...       E001.. IF PASS EQL 3$ BEGIN
1150...     EP3..
1152...        PASS=PASS+1$
1156...         BLK(5) = ''5200''$  I5 = ''6777''$
1164...         GO SCM2$
1165...       END$
1165... EP4..      COMMENT SUCCESSFUL COMPLETION $
1170...              IF I7 NEQ ''3377'' $ TAPE(''704'',1,7)$
1176...         I1 = PROGNUM$
1200...         I2=LOCNO$
1202...         RDC(1,0,1,''230'')$
1204...         DO ''6400''$
1205...      COMMENT EXECUTE A ROUTINE IN QTR 1 WHICH MOVES
1205...         ASSEMBLED PROGRAM TO ITS DESIRED STACK LOCATION$
1205...
1205... BLINK..  I6I = BLNKS(0) $
1207...       I4=I6$ GO BL1$
1212...   BL2..
1217...       M(I6I)=M(I1)+I4-I6-1$
1222...   BL1.. SCAN($$BL2,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(1)-READF(0)$
1245...            REPEAT BLOCN $
1253...            BEGIN
1255...               RELOC= READ(0)$
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$
```

```
      *-SPACE
   ...    COMMENT THIS IS THE LOSS MONITOR  $
   ...
   ...
   ...    EXTERNAL PROCEDURE INCHAR( DMY )$
   ...    EXTERNAL PROCEDURE TYPE(HALF WORD CODE) $
   ...    EXTERNAL PROCEDURE TYPEN( NUMBER ) $
   ...
   ...
   ...
   ...    ARRAY DOTS(2)=(''3636,7304''),
   ...    PROG(4) = (''7334'','PROG$'),
   ...    NOSHDR(8) = (''7334'','NO SUCH PROG$'),
   ...       BADTXT(8) = (''7334'' ,'NO SUCH TEXT$'),
   ...    PARAMS(4) = ('PARAMS$'),
   ...        DEL(3) = (''3434'' ,'XX$'),
   ...       BKUN (7) =( ''7334'','BOOK UNDEF$') ,
   ...    XECU (3) = (''7334'','RUN$'),
   ...          IXHED(15)=(''7373'','NAME/LENGTH/BLOCK NO.',''73'','$$') ,
   ...       NWRMSG(3)=(''7334,3673'','$$'),
   ...       EMSG(4)= (''7334'' , 'ERROR$')   $
   ...
   ...    GLOBAL SGN$
   ...    GLOBAL QUOTE $
   ...     GLOBAL EMSG() $
   ... GLOBAL GETLINE , VAL          $        GLOBAL DEL,NXTCHAR  $
   ...
   ... PROCEDURE  BTWN(X,Y $$ TRU) $
 11... BEGIN
 24...    IF (X LEQ M(I4)) AND ( M(I4) LEQ Y)  $ GO TRU $
 27... END BTWN() $
 27...
 33... PROCEDURE WRITE($LOC) $
 42... BEGIN
 42...    III = LOC $
 44...    NEXT..
 50...       TYPE(MH(I1)) $
 50...    IF MH(III) NEQ '$' $ GO NEXT $
 55... END WRITE() $
 55...
 61... SUBROUTINE GETLINE $
 65... BEGIN
 65...    GETTOP.. I4 =''3300''  $
 67...        TYPE(''73'')$
 71...    GETNXT..    M(I4I) = INCHAR(0) $
 74...       BTWN(0,''73'' $$ GETCONT ) $
100...        WRITE($DEL(1))$
102...        GO GETTOP$
103...         GETCONT..  BTWN(0,''72''$$ GETNXT ) $
107...          RETURN $
110... END GETLINE $
110...
110... SUBROUTINE NXTCHAR $
114... BEGIN
114...    IF M(I4) EQL ''73''$
117...    BEGIN
117...       ENTER GETLINE $
```

```
122...        I4 = ''3300'' $
124...     END $
124...     I4=I4+1 $
130... RETURN ENC NXTCHAR  $
131...
131... PROCEDURE SCAN ( $$ IDENT , NMBR , STR , SEMIC , QUEST , SLASH ) $
140... BEGIN
140...     SGN =1$
143...         GLOBAL TOP$
143...     SCTOP..
147...        BTWN('-','-'$$MINUS)$
147...        BTWN (     'A' , 'Z' $$ CHAR ) $
153...        BTWN (     '0' , '7' $$ DIG ) $
157...        BTWN (     5 , 5    $$ STR1 ) $
163...        BTWN (     ''33'', ''33'' $$ SEM1 ) $
167...        BTWN( '/' , '/' $$ SLSH1 ) $
173...        BTWN( ''37'',''37'' $$ QUEST1 ) $
177...         BTWN(''73'',''73''$$SCSKP)$
203...         BTWN(' ',' '$$SCSKP)$
207...         WRITE($EMSG(1))$
211...         GO TOP$
211...
212...        MINUS.. SGN = -1$
215...         SCSKP..
215...         ENTER NXTCHAR$
216...         GO SCTOP$
217...     CHAR..
217...        I5=I4 $
221...        ENTER NXTCHAR $
222...        VAL =  64M(I5)+M(I4) $
227...     CHAR1..  BTWN('A' ,'Z' $$ CHAR2 ) $
233...        BTWN ('0','9' $$ CHAR2 ) $
237...        GO IDENT $
240...     CHAR2..
240...        ENTER NXTCHAR $
241...        GO CHAR1 $
242...     DIG..     VAL = 0 $
244...     DIG2..    BTWN ('0' , '7' $$ DIG1 ) $
250...        GO NMBR $
251...     DIG1.. VAL = 8VAL + ABS (M(I4) - '0').SGN$
267...      ENTER NXTCHAR $
270...        GO DIG2 $
271...     STR1..    ENTER  NXTCHAR $
272...        BTWN (' ' , ' ' $$ STR1 ) $
276...        BTWN (''73'', ''73'' $$ STR1 ) $
302...        QUOTE =  M(I4) $
305...        GO STR $
306...     SEM1..    ENTER NXTCHAR  $
307...        GO SEMIC $
310...     SLSH1..
311...        ENTER NXTCHAR $ GO SLASH $
312...     QUEST1..
313...        ENTER NXTCHAR $ GO QUEST $
314... END SCAN()  $
314...
332... PROCEDURE ADD(ITEM,CONTROL) $
342... BEGIN
342...     M(I2I)= CONTROL $
344...     M(I2I) = ITEM $
347... END ADD() $
```

```
347...
347...
350...  PROCEDURE SKPDOL( $$ FINAL )$
357...  BEGIN
357...       L1..
357...          IF MH(I2) NEQ '$' $
361...         BEGIN        I2=I2+''4000''$  GO L1  END $
370...          IF MH(I2I) EQL '$'  $ GO FINAL $
374...             IF MH(I2) EQL ''73''$ I2 = I2 + ''4000''$
404...   END SKPDOL() $
404...
404...
404...
410... TOP..
412...    WRITE ( $PROG(1)) $
412...    WRITE ($ DOTS(1)) $
414...    I4 = ''3300'' $    M(I4)=''73'' $
421...    OVER=0$
424...    SCAN ( $$ PNAM , PNUM , ERR1 , ERR1 , TYPRS , ERR1 ) $
433...
433...    PNAM..
433...       COMMENT USER HAS TYPED IN A PROGRAM NAME , FIND STACK
433...       NUMBER (N) AND GIVE HIM A DESCRIPTIVE MESSAGE $
435...       I2 = ''2401''$ N = 0$
440...    PNAM1..  I1 = I5 $
442...       N = N + SGN$ COMMENT ALLOW NEGATIVE PROGRAM NAMES$
446...    PNAM2..
446...       IF MH(I2) EQL '$' $
450...          EITHER IF I1 EQL I4 $
456...          BEGIN
462...             SKPDOL($$ PNAM3) $
462...             PNAM3..  I6 = I2 $   GO GETPS $
465...          END $
465...          OTHERWISE $ GO  PNAM4 $
467...       IF MH(I2) EQL M(I1)$
472...       BEGIN
500...          I1=I1+1$    I2=I2+''4000'' $
503...          GO PNAM2 $
504...       END$
504...    PNAM4..  SKPDOL($$ PNAM5) $
506...             SKPDOL($$ PNAM5) $   GO PNAM1  $
510...
511...    PNAM5..  WRITE($ NOSHPR(1)) $ GO TOP $
513...
514...    PNUM..   COMMENT OVERLAY AN UN-DESCRIBED PROGRAM $
514...       N = VAL $
517...       I6I = PARAMS(1) $
521...       GO GETPS $
521...
522...    ERR1..
524...       WRITE($EMSG(1)) $
524...       GO TOP $
524...
525...    TYPRS..
525...       COMMENT  LIST ALL PROGRAMS   $
525...       I2 = ''2401''$
527...    TYPRS2..
527...       IF SNS 1 $ GO TOP $ COMMENT AVOID EXTRA OUTPUT $
533...       TYPE(''73'') $
533...       WRITE ($ M(I2)) $
```

```
540...        SKPDOL($$ TOP) $  SKPDOL($$ TOP) $
544...        GO TYPRS2 $
544...
544...
545... GETPS..
545...     COMMENT GET PARAMETERS TO PROGRAM AND PUT THEM ON BUFFER  $
547...     TYPE(''73'')$
547...     WRITE($M(I6)) $
554...     WRITE( $DOTS(1)) $
556...         BOOK = -1$
562...        I2 = POINTER + ''4000'' $
562...
566...        NXTITM..
566...     I4=''3300''$
570...         SCAN( $$ TNAM,NM, ST, SM, TYPIX, SLSH) $
577...
577...
577...     NM..
605...        ADD(VAL,''3776'') $
605...        GO NXTITM $
605...
606...        ST..
606...           I3=I2=I2+1$
614...        ST1..
614...           ENTER NXTCHAR $
615...      IF M(I4) EQL QUOTE $
621...           BEGIN
621...              ENTER NXTCHAR $
624...              MH(I2I) = ''74'' $
627...              I2 = BTSET( ''4000'', I2 ) $
634...              M(I3) = I3 - I2 $
640...              GO NXTITM $
641...           END $
641...           MH(I2I) = M(I4) $
643...           GO ST1$
643...
643...
644...     TNAM..   COMMENT  FIND THIS  NAME IN CURRENT BOOK INDEX
644...        AND PUT THE BLOCK NUMBER AND LENGTH ON BUFFER  $
646...              IF BOOK LSS 0$ (BKUNERR.. WRITE($BKUN(1))$ GO NXTITH  )$
654...              I7=''7377'' $
656...     TNAM1..
656...        I1 = I5 - 1 $
662...              IF MH(I7I) EQL 0$
663...        BEGIN
667...           WRITE($BADTXT(1)) $  COMMENT NOT IN THIS INDEX $
667...           GO NXTITM $
670...        END $
670...             I7N=I7+ROTR(1,MH(I7)+2)$
676...             REPEAT MH(I7) $
703...                 IF M(I1I) NEQ MH(I7I) $ EXIT NOFND $
713...     IF I1 NEQ I4-1$ GO NOFND$
722...        L = MH(I7I) $
725...             ADD(MH(I7I)+BOOK,''3776'' )$
733...        ADD( L , ''3776'') $
741...        GO NXTITM $
742...     NOFND..  I7=I7N $    GO TNAM1 $
744...
745...     SLSH..
754...         SCAN ( $$ ERR2 , BKN , ERR2 , ERR2 , ERR2 , ERR2 ) $
```

```
 754...     BKN..  COMMENT BOOK DEFINITION $
 754...            BOOK=64BTCLR(''7770'',VAL)$
 762...            RDC(0,1,7,BOOK)$
 770...     GO NXTITM $
 770...
 771...     ERR2..
 773...        WRITE($EMSG(1)) $
 773...        GO GETPS $
 773...
 773...
 774...    TYPIX.. COMMENT TYPE CURRENT INDEX $
 776...    IF BOOK LSS 0$ GO BKUNERR $
1002...    WRITE($IXHED(1))$
1004...    I7=''7377''$
1006...    UNTIL MH(I7I) EQL 0$
1007...    BEGIN
1011...      IF SNS 1$ GO GETPS $
1013...      TYPE(''73'')$
1015...      REPEAT MH(I7)$ TYPE(MH(I7I))$
1027...      TYPE(' ')$
1031...      TYPEN(MH(I7I))$
1035...      TYPE(' ')$
1037...      TYPEN(MH(I7I))$
1043...    END$
1044...     TYPE(''73'')$
1046...
1046...     GO GETPS $
1046...
1047...        SM..
1047...
1047...      COMMENT   SEMI-COLON 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(I2I)=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...         END$
1140...        TYPE(''73'')$
1142...        IF M(I2I) LSS 0  $   GO  ALPH $
1147...        REPEAT -BTSET(''7000'',M(I2)) $
```

```
1157...        BEGIN    COMMENT TYPE NUMBERS     $
1163...         TYPEN(M(I2I))$  TYPE(' ')$
1165...        END$
1166...        GO NEWITEM $
1166...
1167...     ALPH..    I1 = I2 + ''4000'' $
1173...        I2= I2 - M(I2) $
1177...        REPEAT -2.M(I1) $
1206...        BEGIN     IF MH(I1I) EQL ''74'' $ EXIT  NEWITEM $
1215...                  TYPE( MH(I1)) $
1221...        END $
1222...        GO NEWITEM $
1222...
1223... FINISH$
```

```
/*BLINK
/*EXT INCHAR
/*EXT TYPE
/*EXT TYPEN
/*CODE 1224 7674
/0000/ 16033 01000 00000 06074 07403 00000 00000 00000
/0010/ 16030 01104 00017 12006 00470 16020 00471 16027
/0020/ 01000 10007 00017 01104 00450 00451 16000 16005
/0030/ 01037 14026 06000 16061 01000 00000 06074 07601
/0040/ 00000 16056 00061 00000 01301 14047 36002 00000
/0050/ 01321 01120 07773 00450 16044 16040 01037 14043
/0060/ 06000 16110 01000 00000 14107 00064 03300 36002
/0070/ 00073 36001 00000 01064 16001 00000 00073 16103
/0100/ 16034 27742 16065 16001 00000 00072 16071 00000
/0110/ 16131 01000 00000 14130 01120 07704 01104 00450
/0120/ 16124 16062 00064 03300 01020 00001 02004 04004
/0130/ 00000 16332 01000 00000 06074 07606 00000 16315
/0140/ 12125 01060 00000 16001 00015 00015 16212 16001
/0150/ 00041 00072 16217 16001 00020 00027 16242 16001
/0160/ 00005 00005 16271 16001 00033 00033 16306 16001
/0170/ 00017 00017 16310 16001 00037 00037 16312 16001
/0200/ 00073 00073 16215 16001 00000 00000 16215 16034
/0210/ 27702 16410 01020 07776 14142 16111 16143 00045
/0220/ 00004 16111 01005 00246 01104 01060 00000 16001
/0230/ 00041 00072 16240 16001 00020 00031 16240 00000
/0240/ 16111 16227 00011 14226 16001 00020 00027 16251
/0250/ 00000 01000 10226 00243 01060 00000 01020 07757
/0260/ 01104 00451 00017 01240 10142 12255 14226 16111
/0270/ 16244 16111 16001 00000 00000 16271 16001 00073
/0300/ 00073 16271 01004 01060 00000 00000 16111 00000
/0310/ 16111 00000 16111 00000 16136 01037 14237 01037
/0320/ 14250 01037 14305 01037 14307 01037 14313 01037
/0330/ 14311 06000 16350 01000 00000 06074 07402 00000
/0340/ 00000 00000 12341 01062 01000 10340 01062 16337
/0350/ 16410 01000 00000 06074 07601 00000 16405 01302
/0360/ 12052 00470 16370 01020 04000 02002 04002 16357
/0370/ 01322 12052 00470 16000 01302 12115 00450 16404
/0400/ 01020 04000 02002 04002 16355 01037 14373 06000
/0410/ 16034 27771 16034 27775 00064 03300 01020 00073
/0420/ 01044 00011 01060 00000 16132 16433 16514 16522
/0430/ 16522 16525 16522 00062 02401 00011 01060 00000
/0440/ 00041 00005 01000 10142 12437 14437 01302 12052
/0450/ 00450 16467 01000 00004 00017 02001 00450 16466
/0460/ 16351 16462 00046 00002 16545 16467 16504 01302
/0470/ 00017 01101 00450 16504 01020 00001 02001 04001
/0500/ 12401 02002 04002 16446 16351 16511 16351 16511
```

```
/0510/ 16440 16034 27700 16410 01000 10226 14437 00066
/0520/ 27745 16545 16034 27702 16410 00062 02401 00461
/0530/ 16410 36002 00073 01000 00002 14537 16034 00000
/0540/ 16351 16410 16351 16410 16527 36002 00073 01000
/0550/ 00006 14553 16034 00000 16034 27775 01020 07776
/0560/ 01060 00000 01020 04000 02155 04002 00064 03300
/0570/ 16132 16644 16577 16606 17047 16774 16745 01000
/0600/ 10226 14603 16333 00000 03776 16566 01020 00001
/0610/ 02002 04002 00043 00002 16111 01000 10304 00017
/0620/ 01104 00450 16641 16111 01020 00074 01362 01000
/0630/ 00002 01620 04000 04002 02002 00017 02003 01043
/0640/ 16566 01004 01362 16614 01000 10561 00450 00471
/0650/ 16654 16034 27733 16566 00067 07377 01020 07776
/0660/ 02005 04001 01327 00450 16670 16034 27751 16566
/0670/ 01307 12630 00301 02007 01060 00000 01307 14702
/0700/ 06123 16713 00000 01327 00017 01121 00470 16712
/0710/ 06046 16742 06146 01020 07776 02004 00017 02001
/0720/ 00450 16742 01327 01060 00000 01327 12561 14731
/0730/ 16333 00000 03776 01000 10724 14737 16333 00000
/0740/ 03776 16566 00047 10675 16656 16132 16771 16754
/0750/ 16771 16771 16771 16771 01000 10226 01560 07770
/0760/ 00246 14561 01120 07000 12561 14767 00710 00000
/0770/ 16566 16034 27702 16545 01000 10561 00470 17002
/1000/ 00451 16651 16034 27711 00067 07377 01327 00470
/1010/ 17044 00461 16545 36002 00073 01307 15021 06123
/1020/ 17027 00000 01327 15025 36002 00000 06146 36002
/1030/ 00000 01327 15034 36003 00000 36002 00000 01327
/1040/ 15042 36003 00000 17006 36002 00073 16545 16333
/1050/ 00000 03776 16333 00000 03776 16333 00000 03776
/1060/ 16333 00000 03776 01000 00155 01062 01000 00002
/1070/ 01560 04000 04155 16034 27730 16034 27775 01000
/1100/ 10437 00470 17105 01000 10437 06343 36002 00073
/1110/ 00043 00155 01003 04002 16034 27706 01000 00002
/1120/ 00450 17126 01000 00003 04155 16410 01020 07776
/1130/ 02003 00017 02002 00450 17140 01003 04003 17112
/1140/ 36002 00073 01022 00470 17147 00451 17167 01002
/1150/ 01620 07000 00017 15156 06123 17166 00000 01022
/1160/ 15162 36003 00000 36002 00000 06146 17126 01020
/1170/ 04000 02002 04001 01102 00017 02002 04002 01101
/1200/ 00241 00017 15205 06123 17222 00000 01321 01120
/1210/ 07703 00450 17215 06046 17126 01301 15220 36002
/1220/ 00000 06146 17126 06336
```

```
/*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

ADDR	DEF	TYPE		ADDR	DEF	TYPE		ADDR	DEF	TYPE	
				...SIMPLE VARIABLES...							
10561	LO	INT	BOOK	24	GL	INT	I4I	10724	LO	INT	L
1	GL	INT	I1	5	GL	INT	I5	10437	LO	INT	N
21	GL	INT	I1I	25	GL	INT	I5I	27700	LO	INT	NOSHPR
2	GL	INT	I2	6	GL	INT	I6	10423	LO	INT	OVER
22	GL	INT	I2I	26	GL	INT	I6I	155	GL	INT	POINTER
3	GL	INT	I3	7	GL	INT	I7	10304	GL	INT	QUOTE
23	GL	INT	I3I	27	GL	INT	I7I	10142	GL	INT	SGN
4	GL	INT	I4	10675	LO	INT	I7N	10226	GL	INT	VAL
				...ARRAYS...							
27750	LO	INT	BADTXT	27710	LO	INT	IXHED	27705	LO	INT	NWRMSG
27732	LO	INT	BKUN	0	GL	INT	M	27744	LO	INT	PARAMS
27741	GL	INT	DEL	0	GL	REAL	MF	27770	LO	INT	PROG
27774	LO	INT	DOTS	0	GL	SPEC	MH	27727	LO	INT	XECU
27701	GL	INT	EMSG	27760	LO	INT	NOSHDR				
				...PROCEDURES AND FUNCTIONS...							
10333	GL	INT	ADD	164	GL	INT	PUT	30002	GL	INT	TYPE
10001	GL	INT	BTWN	10132	GL	INT	SCAN	30003	GL	INT	TYPEN
214	GL	INT	GET	10351	GL	INT	SKPDOL	10034	GL	INT	WRITE
30001	GL	INT	INCHAR								
				...LABELS...							
11167	LO		ALPH	11112	LO		NEWR	10514	LO		PNUM
10754	LO		BKN	10577	LO		NM	10745	LO		SLSH
10651	LO		BKUNERR	10742	LO		NOFND	11047	LO		SM
10522	LO		ERR1	10111	GL		NXTCHAR	10606	LO		ST
10771	LO		ERR2	10566	LO		NXTITM	10614	LO		ST1
10103	LO		GETCONT	10433	LO		PNAM	10644	LO		TNAM
10062	GL		GETLINE	10440	LO		PNAM1	10656	LO		TNAM1
10071	LO		GETNXT	10446	LO		PNAM2	10410	LO		TOP
10545	LO		GETPS	10462	LO		PNAM3	10774	LO		TYPIX
10065	LO		GETTOP	10504	LO		PNAM4	10525	LO		TYPRS
11126	LO		NEWITEM	10511	LO		PNAM5	10527	LO		TYPRS2

```
                        SYMBOLS IN PROCEDURE/FUNCTION 'BTWN'
                              ...SIMPLE VARIABLES...
10006  LO   INT     X             10007  LO   INT     Y
                                    ...LABELS...
       NA           TRU

                        SYMBOLS IN PROCEDURE/FUNCTION 'WRITE'
                              ...SIMPLE VARIABLES...
       NA   INT     LOC
                                    ...LABELS...
10044  LO           NEXT

                        SYMBOLS IN PROCEDURE/FUNCTION 'SCAN'
                                    ...LABELS...
10217  LO           CHAR          10212  LO           MINUS
10227  LO           CHAR1                NA           NMBR
10240  LO           CHAR2                NA           QUEST
10242  LO           DIG           10312  LO           QUEST1
10251  LO           DIG1          10215  LO           SCSKP
10244  LO           DIG2          10143  LO           SCTOP
       NA           IDENT         10306  LO           SEM1

                              NA           SEMIC
                              NA           SLASH
                       10310  LO           SLSH1
                              NA           STR
                       10271  LO           STR1
                       10410  GL           TOP

                        SYMBOLS IN PROCEDURE/FUNCTION 'ADD'
                              ...SIMPLE VARIABLES...
10341  LO   INT     CONTROL       10340  LO   INT     ITEM

                        SYMBOLS IN PROCEDURE/FUNCTION 'SKPDOL'
                                    ...LABELS...

   NA          FINAL                    10357  LO          L1

     EXECUTION, NO ERRORS FOUND
```

```
                STANFORD BLINK COMPILER - - VERSION OF 20 MAY 65
      COMMENT  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'','PROG$'),
   ...     NOSHDR(8) = (''7334'','NO SUCH PROG$'),
   ...        BADTXT(8) = (''7334'' ,'NO SUCH TEXT$'),
   ...     PARAMS(4) = ('PARAMS$'),
   ...         DEL(3) = (''3434'' ,'XX$'),
   ...        BKUN (7) =( ''7334'','BOOK UNDEF$') ,
   ...     XECU (3) = (''7334'','RUN$'),
   ...           IXHED(15)=(''7373'','NAME/LENGTH/BLOCK NO.',''73'','$$') ,
   ...        NWRMSG(3)=(''7334,3673'','$$'),
   ...        EMSG(4)= (''7334'' , 'ERROR$')    $
   ...
   ...     GLOBAL QUOTE $
   ...       GLOBAL SGN$
   ...      GLOBAL EMSG() $
   ... GLOBAL GETLINE , VAL            $         GLOBAL DEL,NXTCHAR  $
   ...
   ... PROCEDURE  BTWN(X,Y $$ TRU) $
 11... BEGIN
 23...     IF (X LEQ MH(I4)) AND (MH(I4) LEQ Y)$  GO TRU$
 26... END BTWN() $
 26...
 32... PROCEDURE WRITE($LOC) $
 41... BEGIN
 41...     III = LOC $
 43...     NEXT..
 47...        TYPE(MH(II)) $
 47...     IF MH(III) NEQ '$' $ GO NEXT $
 54... END WRITE() $
 54...
 60... SUBROUTINE GETLINE $
 64... BEGIN
 64...     GETTOP.. I4 = ''3340''$
 66...         TYPE(''73'')$
 70...     GETNXT.. MH(I4I) = INCHAR(0)$
 73...        BTWN(0,''73'' $$ GETCONT ) $
 77...         WRITE($DEL(1))$
101...         GO GETTOP$
102...          GETCONT..  BTWN(0,''72''$$ GETNXT ) $
106...           RETURN $
107... END GETLINE $
107...
107... SUBROUTINE NXTCHAR $
113... BEGIN
113...     IF MH(I4) EQL ''73''$
```

```
116...     BEGIN
116...        ENTER GETLINE $
121...        I4 =''3340''$
123...     END $
123...     I4 = I4 + ''4000''$
127... RETURN END NXTCHAR  $
130...
130... PROCEDURE SCAN ( $$ IDENT , NMBR , STR , SEMIC , QUEST , SLASH ) $
137... BEGIN
137...           SGN = 1$
143...         GLOBAL TOP$
143...     SCTOP..
147...         BTWN('-','-'$$MINUS)$
147...        BTWN (     'A' , 'Z' $$ CHAR ) $
153...        BTWN (     '0' , '7' $$ DIG ) $
157...        BTWN (     5 , 5    $$ STR1 ) $
163...        BTWN (     ''33'', ''33'' $$ SEM1 ) $
167...        BTWN( '/' , '/' $$ SLSH1 ) $
173...        BTWN( ''37'',''37'' $$ QUEST1 ) $
177...         BTWN(''73'',''73''$$SCSKP)$
203...         BTWN(' ',' '$$SCSKP)$
207...         WRITE($EMSG(1))$
211...         GO TOP$
212...        MINUS.. SGN = -1$
212...
215...         SCSKP..
215...         ENTER NXTCHAR$
216...         GO SCTOP$
217...     CHAR..
217...        I5=I4 $
221...        ENTER NXTCHAR $
222...     VAL = 64MH(I5) + MH(I4)$
232...     CHAR1..  BTWN('A' ,'Z' $$ CHAR2 ) $
236...        BTWN ('0','9' $$ CHAR2 ) $
242...        GO IDENT $
243...     CHAR2..
243...        ENTER NXTCHAR $
244...        GO CHAR1 $
245...     DIG..     VAL = 0 $
247...     DIG2..    BTWN ('0' , '7' $$ DIG1 ) $
253...        GO NMBR $
254...     DIG1.. VAL = 8VAL + ABS (MH(I4) - '0').SGN$
271...      ENTER NXTCHAR $
272...        GO DIG2 $
273...     STR1..    ENTER  NXTCHAR $
274...        BTWN (' ' , ' ' $$ STR1 ) $
300...        BTWN (''73'', ''73'' $$ STR1 ) $
304...        QUOTE = MH(I4)$
307...        GO STR $
310...     SEM1..    ENTER NXTCHAR  $
311...        GO SEMIC $
312...     SLSH1..
313...        ENTER NXTCHAR $ GO SLASH $
314...     QUEST1..
315...        ENTER NXTCHAR $ GO QUEST $
316... END SCAN()  $
316...
334... PROCEDURE ADD(ITEM,CONTROL) $
344... BEGIN
344...     M(I2I)= CONTROL $
```

```
346...     M(I2I) = ITEM $
351... END ADD() $
351...
351...
352...  PROCEDURE SKPDOL( $$ FINAL )$
361...  BEGIN
361...       L1..
361...          IF MH(I2) NEQ '$' $
363...         BEGIN        I2=I2+''4000''$  GO L1  END $
372...          IF MH(I2I) EQL '$'  $ GO FINAL $
376...            IF MH(I2) EQL ' '$ I2=I2+''4000''$
405...   END SKPDOL() $
405...
405...
405...
411...          TYPE(''73'')$
413...
413...          COMMENT  TYPE TOP RECORD$
413...       I3 = POINTER $
415...    NEWR..    I2 = M(I3)  $
417...            WRITE($NWRMSG(1))$
421...          IF I2 EQL 0 $
423...          BEGIN  POINTER = I3$  GO TOP  END $
431...    NEWITEM..
431...       IF I2 EQL I3-1$
436...        BEGIN
440...            I3=M(I3)$
442...         GO TOP1$
443...        END$
443...        TYPE(''73'')$
445...       IF M(I2I) LSS 0  $   GO  ALPH $
452...       REPEAT -BTSET(''7000'',M(I2)) $
462...       BEGIN    COMMENT TYPE NUMBERS     $
466...       TYPEN(M(I2I))$  TYPE(' ')$
470...       END$
471...       GO NEWITEM $
471...
472...    ALPH..   I1 = I2 + ''4000'' $
476...       I2= I2 - M(I2) $
502...       REPEAT -2.M(I1) $
511...       BEGIN    IF MH(I1I) EQL ''74'' $ EXIT  NEWITEM $
520...                TYPE( MH(I1)) $
524...       END $
525...       GO NEWITEM $
525...
525...
525...
526...     TOP1.. POINTER = I3$
531... TOP..
531... GETPS..
531...    COMMENT GET PARAMETERS TO PROGRAM AND PUT THEM ON BUFFER  $
533...    TYPE(''73'')$
533...    WRITE( $DOTS(1)) $
535...        BOOK = -1$
541...       I2 = POINTER + ''4000'' $
541...
545...       NXTITM..
547...    I4 = ''3340''$  MH(I4) =''73''$
552...        SCAN( $$ TNAM,NM, ST, SM, TYPIX, SLSH) $
561...
```

```
561...
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 = BTSET( ''4000'', I2 ) $
615...              M(I3) = I3 - I2 $
621...              GO NXTITM $
622...           END $
622...          MH(I2I) = MH(I4)$
624...           GO ST1$
624...
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...              I7=''7377'' $
637...     TNAM1..
637...        I1 = I5 - ''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(I1I) 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..  I7=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...
753...
754...     ERR1..
754...     ERR2..
756...        WRITE($EMSG(1)) $
756...        GO GETPS $
756...
756...
757...    TYPIX.. COMMENT TYPE CURRENT INDEX $
761...    IF BOOK LSS 0$ GO BKUNERR $
765...    WRITE($IXHED(1))$
767...    I7=''7377''$
```

```
 771...    UNTIL MH(I7I) EQL 0$
 772...    BEGIN
 774...      IF SNS 1$ GO GETPS $
 776...      TYPE(''73'')$
1000...      REPEAT MH(I7)$ TYPE(MH(I7I))$
1012...      TYPE(' ')$
1014...      TYPEN(MH(I7I))$
1020...      TYPE(' ')$
1022...      TYPEN(MH(I7I))$
1026...    END$
1027...     TYPE(''73'')$
1031...
1031...     GO GETPS $
1031...
1032...        SM..
1032...
1035...           ADD(0,''3776'')$
1035...           ADD(0,''3776'')$
1040...           ADD(0,''3776'')$
1043...           ADD(0,''3776'')$
1046...           M(I2I)=POINTER $
1051...           POINTER = BTCLR(''4000'', I2) $
1056...           WRITE($XECU(1))$  WRITE($DOTS(1))$
1062...
1062... FINISH$
```

```
...

   *-SPACE
... COMMENT  THIS PROGRAM DEFINES AND ERASES TEXTS.
...        ITS  INPUT IS A SINGLE ALPHABTIC 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 BADOP$
...
...      STRING
...               FULL(12)=('BOOK CAPACITY EXCEEDED'),
...               BDOPR(10)=('ILLEGAL OPERATION')$
...     ARRAY AVAIL(0..66),CHAR(10)    $
...
...
...
... PROCEDURE GETEL($$ALPHA,NUMBER)$
 7... BEGIN
 7...
33... FUNCTION BTWN(X,A,Y)= (X LEQ A) AND (A LEQ Y) $
33...
43...   GCH..
46...    EITHER IF MH(I6) EQL ''74''$  RETURN$
51...    OR IF BTWN('A',MH(I6),'Z')$
60...    BEGIN
64...             I1I=CHAR(1)$ I2=I1$
66...       MH(I1I)=MH(I6)$
70...             UNTIL NOT(BTWN('A',MH(I6I),'Z') OR BTWN('0',MH(I6),'9'))$
106...               MH(I1I)=MH(I6)$
113...            MH(I2)=2(I1-I2)$
121...            MH(I1I)=0$
123...       GO ALPHA $
124...    END$
124...    OR IF BTWN('0',MH(I6),'7')$
133...    BEGIN
140...            CHAR(1)=ABS(MH(I6)-'0')$
144...       UNTIL NOT BTWN('0',MH(I6I),'7')$
152...          CHAR(1)=CHAR(1)8+ABS(MH(I6)-'0')$
171...       GO NUMBER$
172...    END$
172...    OTHERWISE$  INCR(''4000'',I6)$
177...    GO GCH$
200... END GETEL()$
200...
206...       SUBROUTINE WRITE$
212...       BEGIN
214...                IF BOOK GEQ 0$ WRC(0,1,7,BOOK) $
226...       RETURN END WRITE $
```

```
227...
227...        PROCEDURE ERITE(STR())$
237...        BEGIN
237...           STRING S(11)=('AT OPERATION NUMBER') $
237...           LIST L(STR(),S(),COUNT) $
255...           PUT($$L)$
257...         OVERLAY 0$
261...        END ERITE() $
261...
261...
262...        PROCEDURE SEARCH($$FOUND)$
271...        BEGIN
271...           I7=''3377''$
273...           I5=''2377''$
275...           REPEAT 256 $ M(I5I)=M(I7I)$
303...     I7=''7377''$
303...
305...              GETEL($$SRCH1,BADOP)$ GO BADOP $
311...     SRCH1..
311...     UNTIL MH(I7I) EQL 0$
312...     BEGIN
314...         I1I=CHAR(1)$
316...         MATCH=0$
321...         I5=I7-(''1000''+''4000'')$
325...         I2=I7+ROTR(1,MH(I7)+2)$
332...         REPEAT MH(I7)$IF MATCH EQL 0$
341...                  MATCH=MH(I1I)-MH(I7I)$
353...         I7=I2$
355...         IF MATCH LSS 0$
357...         BEGIN COMMENT NOT PRESENT$
357...             I7=I5+''1000''$
366...             RETURN$
367...         END$
367...
367...         IF MATCH EQL 0$
371...             IF MH(I1I) EQL 0$ GO FOUND$
376...        COMMENT KEEP LOOKING $
376...     END$
376...
377...           COMMENT NOT PRESENT$
403...           I5=INCR(-''4000'',I7)-''1000''$
406...        END SEARCH() $
406...
412...        SUBROUTINE FINIX $
416...        BEGIN
416...           UNTIL MH(I7I) EQL 0 $
417...           BEGIN MH(I5I)=MH(I7) $
423...             REPEAT MH(I7)+2 $ MH(I5I)=MH(I7I) $
434...           END$
435...           UNTIL I5 EQL ''6777'' $ MH(I5I)=0 $
442...
445...      COMMENT   NOW MOVE NEW INDEX TO QUARTER 7$
447...           I7=''3377''$  I5=''2377''$
451...      REPEAT 256$ M(I7I)=M(I5I)$
454...
454...
457...        RETURN END FINIX $
460...
460...     BOOK = -1 $     COUNT=0$
465...     I6=M(POINTER)+2$ IF M(I6-1) GTR 0$ GO BADOP$
```

```
 504...      POINTER = M(POINTER ) $
 512... OPR..  COUNT=COUNT+1$
 520...        GETEL($$ALPH,BADOP)$
 523... NOMO..    ENTER WRITE$
 524...           RETURN $
 525... ALPH..      IF MH(I2I) EQL 'B' $
 530...           BEGIN
 535...              GETEL($$BADOP,NWBK)$ GO BADOP $
 536...           NWBK..
 536...              ENTER WRITE $
 537...              BOOK=BTCLR(''7770'',CHAR(1))64 $
 545...              RDC(0,1,7,  BOOK) $
 552...              I4I=AVAIL(0)$      BASE=I4$
 560...              REPEAT 64$  M(I4I)=0$
 566...              I7=''7377''$
 570...            UNTIL MH(I7I) EQL 0$
 571...              BEGIN
 573...                     I7=I7+ROTR(1,MH(I7))$
 577...                     SIZE=MH(I7I)$
 602...                     I4=BASE+MH(I7I)-1$
 606...                REPEAT SIZE $ M(I4I)=1$
 620...              END$
 621...              GO OPR $
 622...           END$
 622...              IF BOOK LSS 0$ GO BADOP $
 630...                 IF MH(I2) EQL 'D' $
 633...           BEGIN
 637...              SEARCH($$BADOP)$
 637...            I1I=CHAR(1)$ MH(I5I)=MH(I1)$
 643...            REPEAT MH(I1)$ MH(I5I)=MH(I1I)$
 653...              GETEL($$BADOP,LGTH)$ GO BADOP $
 657...           LGTH..
 657...            MH(I5I)=CHAR(1)$
 662...              I4=BASE$
 664...              ZERS=0$
 667...        ITER..
 674...              IF I4 EQL BASE + 63 $ ERITE(FULL())$
 701...              EITHER IF M(I4I) EQL 1 $  ZERS=0 $
 710...              OTHERWISE$  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$
 746...              ENTER  FINIX $
 747...              GO OPR $
 750...           END$
 750...                 IF MH(I2) EQL 'E'$
 753...           BEGIN $
 753...              SEARCH($$ERFND)$
 757...           BADOP..  ERITE(BDOPR())$
 762...           ERFND..
 764...                      I4=BASE+MH(I7)-1$
 766...                REPEAT MH(I7-''4000'')$ M(I4I)=0 $
1002...              ENTER  FINIX $
1003...              GO OPR $
1004...           END$
1004...           GO BADOP $
1004...
1005...    FINISH$
```

```
...
...     COMMENT  THIS PROGRAM LOADS A BLINK TEXT  $
...      EXTERNAL PROCEDURE INREC (N$MINL,MAXV$ERRCAS)$
...      EXTERNAL SUBROUTINE CONVRT$
...     ARRAY BUFF(0..10)$
...     GLOBAL B,BMAX,ERR,SSAV,NL  $
...     GLOBAL SENDN , PARS $
...     GLOBAL BSAVE$
   *-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 S1 (7)=('TAPE ERRORS')$
 20...     LIST L1(S1())$
 26...        RDC(0,0,6,''170'')$
 30...     M(''21'')=BSAVE$
 33...     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...     I1=SSAV$
 63...     IF I1 EQL ''6377''$
 66...     BEGIN
 74...         WRC(0,1,4,INCR(1,B))$
101...         I1=0$  I1=-I1$
107...         IF B EQL BMAX$ QUIT( FULL())$
120...     END$
120...     MH(I1I)=N$ SSAV=I1$
126...  END SEND()$
126...
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($$L)$
166... BEGIN
166...    ENTER SENDN$
167...    IF MH(I7I) EQL ' '$ GO NL$
172...    IF MH(I7) EQL ''73''$ GO NL$
177...    GO L$
200... END TSTEOL()$
200...
200...
204...       WRC(0,0,6,''170'')$
206...        BSAVE=M(''21'')$  I1I=BUFF(0)$  M(''21'')=I1$
215...    SSAV  = ''6000''$
217...    GET ($$PARS)$
222...    IF BLK LEQ 0$
224...       QUIT(FULL())$
233...    B=BLK-1$ BMAX=B+LENGTHS
237...
243... NR..
243...     ERR1 = 0$
246...     INREC(1$M(''2401''),I5$ERRCAS)$
253...     CONV..  ENTER CONVRT$
254...    I6=''6400''$
256...    CKD=0$
256...
261... NL..
261...    IF SNS 1$ QUIT( SNS1())$
266...    I7=I6$
270... NC..
273...    IF MH(I6I) EQL ''74''$ GO NR$
275...    IF MH(I6) NEQ ''73''$ GO NC$
301...    IF MH(I7I) NEQ ' '$ GO NL$
304...    IF MH(I7I) NEQ '/'$ GO NL$
311...    IF NOT BLFND$
313...        BEGIN BLFND=(MH(I7+1) EQL 'B')$ GO NL$ END $
335...
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...    END$
362...    IF MH(I7I) EQL '*'$ GO PSEUDOS$
367...    IF VRBFND$
371...    BEGIN
375...        NXTV..  TSTEOL($$NXTV)$
375...    END$
375...    COMMENT CODE CASE, SKIP ADDRESS$
375...    I7=I7+3$
401...    NEXTC.. SEND(MH(I7)-'0')$
407...        I7=I7+''4000''$
413...        TSTEOL($$NEXTC)$
415... PSEUDOS..
415...    EITHER IF MH(I7I) EQL 'E'$
420...    BEGIN COMMENT 'EXT' HAS APPEARED$
420...        I7=I7+''4001''$
426...        UNTIL MH(I7I) EQL ' '$ SEND(MH(I7))$
436...        SEND(' ')$
440...    END$
```

```
440...     OR IF MH(I7) EQL 'C'$
444...     BEGIN COMMENT 'CODE'$
450...         SEND(''73'')$ COMMENT TERMINATE SYMBOLIC PART$
450...         I7=I7+''4002''$
454...         ENTER SENDN$
455...         I7=I7+''4000''$
461...         ENTER SENDN$
462...     END$
462...     OR IF MH(I7) EQL 'V'$
466...     BEGIN COMMENT 'VARB'$
466...         VRBFND=1$
473...     END$
473...     OR IF MH(I7) EQL 'F'$
477...     BEGIN COMMENT 'FINISH'$
503...         SEND(0)$
503...     SEND(0)$
505...         WRC(0,1,4,INCR(1,B))$
515...         RDC(0,1,4,BLK)$
523...         MH(''2000'')=B-BLK+1$
532...         MH(''6000'')='B'$
536...         WRC(0,1,4,BLK)$
544...         QUIT( OK())$
547...     END$
547...     OTHERWISE$ ERR=1$
553...     GO NL$
554...      ERRCAS..  ERR1 = 1$ GO CONV$
560...     LIST PARS (BLK,LENGTH)$
573... FINISH$
```