A description of a Turing Machine simulator, programmed on the CDC 3300 Digital Computer, using the COMPASS programming languages Version 2.0 of the OS3 system. The simulator will build a quintuple list from the input of a Turing Machine, then execute the machine as dictated by the data to be analyzed. The printed results will give the answer obtained by the use of the machine. The simulator will also allow a Turing Machine to be saved, retrieved, corrected or destroyed on a file. This simulator is unique in that it enables a user to design a Turing Machine to write and execute a second Turing Machine.
A Turing Automaton Simulator Program

by

James Theodore Kolbet

A THESIS

submitted to

Oregon State University

in partial fulfillment of the requirements for the degree of

Master of Science

June 1971
APPROVED:

Redacted for Privacy

Professor of Mathematics in charge of major

Redacted for Privacy

Chairman of Department of Mathematics

Redacted for Privacy

Dean of Graduate School

Date Thesis is Presented April 12, 1971

Typed by Carolyn Irving for James Theodore Kolbet
## TABLE OF CONTENTS

**INTRODUCTION**  
1

**THE MAIN ROUTINE**  
4

- Input and Output  
4
- Heading, Date and Time  
5
- Initialization  
7
- Pivoting Between FETCH, EXEC and FILES  
7
- Processing Builder Errors  
10
- The Turing Machine Written by Another Machine  
12
- STOP2, The Clean-up  
14

**THE FETCH ROUTINE**  
15

- FETCH and the Building Process  
16
- Character Analysis  
20
- Secondary Support Routines  
24

**THE EXEC ROUTINE**  
26

- Processing the Parameters  
26
- The Execution  
30
- Supporting Routines  
32

**THE FILES ROUTINE**  
38

- The Four Major Functions  
39
- Supporting Routines  
44

**EPILOGUE**  
46

**APPENDICES**  

- Appendix I  Test Cases  
47
- Appendix II  Using the Turing Machine Simulator  
68
- Appendix III  Flow Charts  
86
- Appendix IV  Program Definitions  
113
- Appendix V  Program Listing  
121
A TURING AUTOMATON SIMULATOR PROGRAM

INTRODUCTION

In designing Turing Machines, it is desirable and often necessary to have a simplified means with which to check and debug them. To facilitate this debugging and checking process, the digital computer becomes an effective and sometimes vital tool. As a problem solving device, as an algorithmic proof or simply as a teaching aid, the designing of the Turing Machine is simplified by a simulation process. The designer, with use of the simulator, can now devote more time to the designing of the Turing Machine, rather than to the tedious and at times frustrating hours needed for the actual testing and debugging processes required by the machine.

This thesis deals with the construction of the Turing Machine simulator *TURM. *TURM is designed for use on the CDC 3300 Digital Computer. It is written in the COMPASS programming language Version 2.0 of the OS3 system, and saved under this system.

*TURM consists of four basic routines: MAIN, FETCH, EXEC, and FILES. Each routine corresponds to the basic needs of a Turing Machine. MAIN prepares the computer for the Turing Machine by initialization of tables, counters, flags and pointers. FETCH prepares the Turing Machine for the computer by building quintuples and storing them into memory. In this respect, FETCH becomes the
Turing Machine builder. EXEC executes the prepared Turing Machine as determined by the data to be analyzed, and outputs any or all of the executed data as desired. In this respect, EXEC becomes the execution phase of the Turing Machine. Finally FILES retains any Turing Machine for further reference, eliminating the need for the builder in further runs. FILES, although not a requirement for the building and execution processes, proves a helpful aid to the User in that it stores the Turing Machine on a file and saves it under the name desired.

The explanation of each of the basic four routines requires also the explanation of the supporting routines within that routine. In many cases, the supporting routines may be referenced throughout the entire program. Therefore to maintain some type of continuity, the supporting routines will be discussed within the routine of the basic four in which they are constructed.

In order for the building and execution of the Turing Machine to have a basis for processing, the User's Turing Machine must be of a specific format. The format required consists of a Turing Machine name; a quintuple list; an Exec card with a list of optional parameters; an Information Part, if so desired; and finally an End of File card followed by either a Logoff card or another Turing Machine of the same format.

The Turing Machine name can be any eight letter or digit
character string in which the first character must be a letter.

The quintuple list may be as long or short as is required for the Turing Machine, but the list must be terminated by the word DONE and each quintuple must be of the required format.

The Exec card must be a card with the word EXEC followed by an optional list of parameters, terminated by the word DONE or the IP-parameter to indicate the end of the parameter list and the beginning of the Information Part.

The Information Part can be of any form as desired by the User, but limited to 4000 characters.

*TURM will process the Turing Machine in the order of occurrence as stated above, unless a file function is desired by the User. The file function will be processed before the execution phase begins and after the building process has been completed.
THE MAIN ROUTINE

The Main routine begins the processing of the Turing Machine. The basic functions of the MAIN routine consist of: a) processing the input and output units; b) processing the Heading, Date and Time for the run; c) initiating the counters, pointers, tables and flags; d) pivoting between the building process in FETCH, the execution process in EXEC, and the files processing in FILES; e) processing of the builder errors; f) saving and preparing for the input of a Turing Machine written by another Turing Machine; and g) cleaning up the storage area of the Turing Machine just processed, and setting up storage for the next Turing Machine to be processed.

Thus the MAIN routine has several and varied functions. Though most are not part of the structure of a Turing Machine, they are helpful in handling the simulation properly and in an orderly fashion. Each function then plays an important role in the processing of the Turing Machine.

Input and Output

Upon summoning *TURM, the input and output units are immediately determined by the *SYSLIB routine called IOLUNS. The IOLUNS routine has a limited use within this program, in that the routine sets the input unit for the Turing Machine to the logical unit number 60,
and sets the output unit for the Turing Machine to the logical unit number 61. Thus the use of *TURM is limited to either Batch or to Teletype. Other units could be used, but such use would require the same format for input as that required for any Turing Machine. Also the output would be of such a form that it would be quite unacceptable for any practical use. Thus the workings of the IOLUNS routine will not be discussed in greater detail.

**Heading, Date and Time**

The Heading for the Turing Machine consists of a storage area for the name of the Turing Machine, the Date of the run and the Time of the run. The Heading is a storage area of 26 computer words. In the first two words a carriage control character and the name of the simulator are stored. Once the name of the Turing Machine has been determined, it is stored in the fourth and fifth words of the Heading. The date is computed by a supporting routine called TDATE and is stored in the twenty-third and twenty-fourth words of the Heading in the form: MM/DD/YY. Finally the time is computed by the supporting routine called TIMER and stored in the twenty-sixth word of the Heading in the form: HHMM. Once the heading has been completed, the program name, title, date and time will upon request be printed at the top of each page (Figure 1).

The TDATE routine loads and converts the date to decimal.
The date is ever present in register 37. Once loaded into memory, the date is separated by section and converted into two digit decimal numbers. As it is converted, it is stored in the location specified for it in the Heading. The month is converted and stored in the form MM, followed by a slash (/). The day is then converted and stored in the form DD. Finally after storing a second slash, the year is converted and stored in the form YY. Upon completion, TDATE returns to MAIN and TIMER is summoned to compute the time.

TIMER computes the hour from the same date word stored in register 37. The hour is computed in the same way as the date. Therefore the hour section of the date is loaded and converted into a two digit decimal number and stored in the Heading in the form HH. The minutes on the other hand are stored in register 22 and are in milliseconds. Thus, the minutes must be converted to the two digit decimal number by converting the milliseconds to minutes than to decimal. Once the minutes are converted, they are stored in the area of Heading specified in the form MM. Upon completion TIMER returns again to the MAIN routine for the next phase of the processing. As processed the Heading will appear on the top of each page used by the builder routine.

<table>
<thead>
<tr>
<th>*TURM</th>
<th>(name)</th>
<th>MM/DD/YY</th>
<th>HHMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 wds.</td>
<td>1 wd.</td>
<td>2 wds.</td>
<td></td>
</tr>
<tr>
<td>17 wds.</td>
<td>2 wds.</td>
<td>1 wd.</td>
<td>1 wd.</td>
</tr>
</tbody>
</table>

Figure 1. The Heading
Initialization

After the Heading has been processed, the counters, tables, flags and pointers for the majority of the program are set to their initial values. The storage area CARD, allocated for the input line, and the area set aside for the title or Turing Machine name are filled with blanks. The pointers, flags and counters are all set to zero. The file name area, file name index and the character storage area are all filled with blanks. The column counter for CARD is set to 72 which after the title is processed will immediately cause a new line or card to be read. The address for the top of usable memory is computed and stored for the input of the quintuple list. And the line counter for the output of the quintuple list is set to zero. The remaining pointers, counters, flags and tables are initialized in the routine in which they are used. Now the MAIN routine is ready to begin the processing of the Turing Machine name.

Pivoting Between FETCH, EXEC and FILES

On the first run of the simulation of the Turing Machine, the title must be processed, and at times saved for further reference. Thus the first input card after the first run need not be a title card. Once the Turing Machine has been processed by the builder, there is no need to have the builder re-process it. MAIN checks for this
situation by checking the first non-blank character of the line for either a letter or a slash, (/). Any other character will be deemed erroneous. If the character is a letter, the string following must be the title and will be treated as such by packing the string left-justified in the two word storage location SYM. The name in SYM is then stored in the title area designated for it in the Heading. A page eject follows, printing the Heading on the top of the new page. The TM flag for the special run of a Turing Machine written by another Turing Machine is then checked to see if the column counter need be changed. The change is necessary to prevent the loss of the first line of data in the second Turing Machine. For the first run the title must be on a line of its own. However this same condition is not feasible for the Turing Machine written by another Turing Machine. The restriction of size for the Information Tape, the type of check used for the determination of the end of the Information Tape, plus the waste of space in the Information Tape make this type of spacing for the title of the special Turing Machine impractical. Once the TM flag has been checked and the necessary changes made, MAIN transfers control to the FETCH routine to begin the actual building process.

If however the first non-blank character was not a letter, but was a slash, the MAIN routine has then been notified that a file function follows. The file function must then be a call for a saved Turing Machine to be used on a specific Information Tape, or a call to
correct a saved Turing Machine, or a call to destroy a saved Turing Machine. The saving of a Turing Machine is not considered, since the saving of the machine would have been accomplished during the first run. The file function is determined in the same way as the title, the function being packed left-justified in the word SYM. The function is then stored in the in the word FSYM. Since it must be known which Turing Machine is needed for the function, the title of the machine must follow the file function. Again the title must begin with a letter. If the first character of the string is not a letter, an erroneous condition has been encountered and an error is indicated. If the character is a letter, the title is processed in the same manner as before and stored in the Heading. The FILES routine is then summoned to execute the desired function. Upon return from FILES, with the exception of the destroy function, the Turing Machine is in memory and is ready for the execution of the machine upon the desired Information Tape. MAIN continues by checking for an Exec card or line. If the word EXEC is discovered by packing the first letter string on the next card, MAIN transfers control to the EXEC routine to execute the Turing Machine as dictated by the Information Part. However, if the letter string is in any way erroneous, the message NO EXEC CARD, NO EXECUTION is printed and MAIN jumps to the MAIN clean-up section STOP2.

If the first non-blank character was neither a letter nor a slash, the erroneous condition is treated as the title and stored in the title
area of the Heading. A page is again ejected, printing the Heading and following it with the message **ILLEGAL TITLE OR FILE CALL**. The MAIN routine then jumps to the MAIN clean-up routine STOP2.

**Processing Builder Errors**

The routines ERROR and ERRPRINT in MAIN may be summoned by the builder routine to process any builder errors. The builder uses ERROR to flag an erroneous element in a quintuple. ERROR in turn uses an error table, ERRTAB, for storing the line number in which the error occurred; for storing the error code, to indicate the type of error encountered; and for storing the quintuple as far as it has been constructed. ERROR also counts the number of errors in the table, allowing a total of 30 errors before printing the message **TOO MANY ERRORS**. If 30 errors have been encountered, ERROR jumps to the MAIN clean-up.section STOP2. The error code not only indicates the type of error encountered, but also the number of elements in the erroneous quintuple which must be ignored to allow the continuation of the building phase of the Turing Machine. By continuing the processing and storing of the legal quintuples, the Turing Machine may be saved and corrected. This machine may also retain the capability of working on the specific Information Tape. Thus it is necessary to ignore the remaining elements of the erroneous quintuple in order to eliminate the entire quintuple. If the erroneous
quintuple can not be corrected by the correction file function, it will not be stored in the quintuple list. The quintuple, if needed by the execution phase, will then be considered missing and therefore correctable. Once the error has been stored in the error table and the quintuple elements have been ignored, ERROR returns to the builder for the continuation of the building process.

ERRPRINT will be summoned when the building process has been completed or when a terminating condition occurs in the builder. ERRPRINT simply converts the coding stored in the error table ERRTAB to an output line and prints the error message. If no errors have been encountered, ERRPRINT simply returns to the calling program. However, if there are errors, ERRPRINT loads the line number stored in the first word of the table, converts it to decimal and stores it in the error message line ERRLINE. The error code is then loaded and converted to an index for the specific error message. Index zero stores the message **ILLEGAL REPLACE**, LABEL in the ERRLINE. Index six stores the message **MULTIPLE DEFINED QUINT** in the ERRLINE. And index twelve stores the message **ILLEGAL QUINT LABEL** in the ERRLINE. The erroneous quintuple is then loaded and converted to a decimal quintuple label followed by the symbol being scanned and stored in the last three words of the ERRLINE. ERRLINE is printed and ERRPRINT repeats the above process for the entire table. When there are no other
errors in the error table, ERRPRINT returns to the calling routine to continue the processing of the Turing Machine.

The Turing Machine Written by Another Machine

MAIN also handles the transferral of data used in the processing of a second Turing Machine written by another Turing Machine. The STM routine in MAIN will save the Information Tape which contains the second Turing Machine; will change the input unit to logical unit four, to enable the second Turing Machine to be accepted in the same format as the first Turing Machine; and will return to the beginning of the MAIN routine to read in the second Turing Machine for processing. It is then required of STM to be summoned by the EXEC routine before the clean-up process can be called. Once summoned, STM searches the Information Tape for the data to be transferred. Upon locating the Information data, STM transfers it to logical unit four 18 words at a time, thus allowing for the same input format required for any Turing Machine. Once the end of the Information data has been reached, a file mark is written onto the file and the file is rewound. STM then changes the input unit to logical unit four, saving the original input unit number for later use. STM sets the needed flags to indicate to MAIN and FETCH that this Turing Machine was constructed by another Turing Machine, then proceeds by storing zeroes in the old quintuple list and transferring control to the beginning
of MAIN to start the initialization process for the input of the second Turing Machine. When FETCH has finished the building process, IPLC, an STM subroutine, is summoned to change the input unit number back to the original unit number.

STM must also eliminate the problem of the internal blank. The external blank, the blank separating character strings on the card or line, must be distinguished from the internal blank, the blank used by a Turing Machine. In the normal input of the Turing Machine, a dollar sign, ($), is used to represent the blank to be scanned on the Information Tape. When writing the second Turing Machine, the dollar sign has already been replaced by the octal representation of a blank. Thus the dollar sign in fact will never appear on the Information Tape. But to represent a blank for the second Turing Machine, a dollar sign is needed. STM then must search each 18 word section of the Information Tape for the second internal representation of the blank. In writing a Turing Machine to write a second Turing Machine, it will be necessary for the blanks used in the second machine to be represented by a period, (.). The period will be used in the second Turing Machine in the same way the dollar sign is used in the first Turing Machine. STM will replace the period with a dollar sign before the 18 words are transferred to the file. The input of the second Turing Machine will then be of the same format as that of the first Turing Machine, enabling the builder to process the machine as if it
were any other Turing Machine.

STOP2, The Clean-up

Finally, MAIN takes care of the clean-up of storage after a Turing Machine has been processed. The clean-up simply continues to read cards of lines until an end of file is read. STOP2 will also store zeroes in the storage area allocated for the quintuple list. By cleaning up the storage area, and by reading cards or lines until the end of file is reached, MAIN sets up memory for the input of another Turing Machine. Thus STOP2 upon finishing the task of cleaning up memory, immediately transfers control back to the beginning of the MAIN routine and the initialization process. The Heading, date and time will not be recomputed, nor will the input and output units, for they are assumed to be the same. However, the title, tables, pointers and counters are recomputed and reset so that *TURM can begin the processing of the next Turing Machine. This process will continue until the Logoff card has been read, which terminates *TURM.
THE FETCH ROUTINE

The main function of the FETCH routine is the building of the quintuples into a workable form and the storing of these quintuples into an accessible area of memory. The building process demands the analysis of each character on a card or line. To facilitate the analysis, several support routines are summoned throughout the building process. Thus the FETCH discussion is divided into three categories: 
a) FETCH and the building process; b) the character analysis; and c) the secondary support routines.

As the Turing Machine requires a specific order for simulation, so does the quintuple. In order that FETCH may properly build the quintuple and the quintuple list, the quintuple itself must be in a specific order to facilitate the building. Therefore the quintuple must consist of five elements in the form: \( Q_i \quad S_i \quad Q'_i \quad S'_i \quad M \).

\( Q_i \) is the quintuple label and must be either a digit string or the word \textit{DONE}. The digit string allowed consists of the decimal numbers between 1 and 1023 inclusively. The digit string when converted to binary will use 12 bits of a storage word called \textit{QUINT}. The word \textit{DONE} is allowed only at the end of the quintuple list and will occupy the entire word. The remaining four elements in this case are not needed.

\( S_i \) is the symbol being scanned in the quintuple label and therefore
may be any single BCD character. The character uses six bits and is logically "or"ed into the storage word QUINT. The quintuple label and the symbol being scanned form the quintuple address.

\( Q'_i \) is the replacement label and like \( Q_i \) must either be a digit string or, in this case, the word STOP. If it is a digit string, it is handled in the same way as \( Q_i \) but is stored in the storage word RSTAT. If it is the word STOP, a zero is substituted for the digit string and stored in the 12 bit area of RSTAT.

\( S'_i \) is the replacement symbol and is handled in the same way as \( S_i \) but in this case is logically "or"ed into RSTAT.

M is the move code which can be any single BCD character preferably a digit. The move code signified either a one cell move to the right, a one cell move to the left, or no move at all. If the move code is a zero or a one, the zero or one will be added to the Information Tape column counter which will accomplish no move or a move right one cell. Any other BCD character will indicate a negative move and a minus one will be added to the column counter which will accomplish a one cell left move on the Information Tape.

**FETCH and the Building Process**

The building process is also divided into three phases. First on summoning FETCH, the primary initiation of the routine takes place. Next FETCH3 prepares, through character analysis, the quintuple
Finally, FETCH2 prepares, through character analysis, the replacement conditions and stores the quintuple in memory for later referencing. The analyzed quintuple is stored in two words of memory allocated for the quintuple list, with the quintuple address in the first word and the replacement conditions in the second word, Figure 2.

The initialization phase, FETCH, simply checks the last character read for a blank. If it is not a blank, the call of INCHAR by FETCH3 is not required to load from a line or card the first non-blank character. If it is a blank, INCHAR needs be summoned to begin the processing of the first quintuple label.

The building phase of the quintuple address, FETCH3, then begins by locating the first non-blank character on a card or line. Since the quintuple must be of a specific order, the character must be a digit indicating a digit string, or a letter indicating the word DONE. If neither a digit nor a letter is present, an error condition has been encountered. Thus the erroneous element and the error code are sent to the ERROR routine in MAIN, and the error is stored in memory. If the character is a digit, the quintuple label is converted to binary by calling DECONV, and stored in QUINT. INCHAR then locates the next non-blank character, and this character is logically "or"ed into the word QUINT forming the quintuple address. At this point, FETCH3 checks an edit return flag to determine if the FETCH3
routine was summoned by the EDITS routine. If the flag is set, FETCH3 will then return to the EDITS routine. If the flag is not set, the quintuple address is compared to each quintuple address in the quintuple list by the SERCH routine. Any multiply defined quintuple discovered by SERCH summons the ERROR routine which notes the condition and ignores the second quintuple of the same address. Upon returning from ERROR, the processing continues by starting at the beginning of FETCH3. If the quintuple address was not in the quintuple list, it is saved in the first word of a two word storage location called STATE.

FETCH2 then begins the analysis of the replacement conditions. INCHAR locates the next non-blank character which must indicate another digit string or the Turing Machine stop condition represented by the word STOP. If neither the digit string nor the letter string is indicated, a third error condition has been encountered and ERROR is again summoned to process this condition. Upon return from ERROR, the quintuple address is stored in the quintuple list. The replacement conditions are replaced by the erroneous stop condition word STOP, and stored after the quintuple address. Finally the process continues by returning to the beginning of FETCH3 for the processing of the next quintuple.

If the character was a letter, the string is packed left-justified by the PSYM routine and stored in the word SYM. If the string is not the word STOP, the same error condition has been encountered.
and will be processed as above. If however it was a STOP, an internal stop condition is stored in the storage word RSTAT, i.e. zeroes are stored in the replacement label area of RSTAT. If it was a digit, the digit string is converted to binary by the DECONV routine and the 12 bit string is stored in RSTAT. INCHAR locates the next non-blank character, the replacement symbol. This symbol is then logically "or"ed into RSTAT. Finally, INCHAR locates the move code also storing it in the word RSTAT. Again the edit return flag is checked, and if it is set, FETCH2 returns to the EDITS routine calling location. If it is not set, the word RSTAT is stored in the second word of the two word storage location STATE. The STORE routine then stores the processed quintuple in STATE into the memory area allocated for the quintuple list.

If the entire quintuple list has been stored into memory and converted by the FETCH3 and the FETCH2 sections of FETCH, and the word DONE has not been encountered, the error condition stated in FETCH3 processes the erroneous condition by use of ERROR. Later on, the missing DONE will be indicated by one of the supporting routines. However, if the DONE has been encountered by FETCH3, the flag for change of input unit is checked, and if needed the input unit is restored to the original unit number. This flag would have

```
<table>
<thead>
<tr>
<th>Not used</th>
<th>Q_i</th>
<th>S_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>Not used</td>
<td>Q'_i</td>
</tr>
</tbody>
</table>
```

Figure 2. The quintuple.
been set only in the case of a Turing Machine's writing and executing
a second Turing Machine. A done flag is set for use in the EDITS
routine. The edit return flag is checked to see if FETCH3 was
originally summoned by the EDITS routine also to indicate to this
routine the end of the corrections. If FETCH3 was not so summoned,
FETCH3 calls ERRPRINT to print a list of any errors that might have
been encountered. Finally, FETCH3 checks the next card or line for
an EXEC or any non-blank, non-letter, non-digit character. If the
word EXEC is encountered, the building process has been completed
and FETCH3 transfers control to the EXEC routine. If the character
is neither a letter nor a digit, the condition may indicate a file func-
tion and thus FETCH3 transfers control to the MAIN routine for the
file check. If neither of the above cases are encountered, the error
condition is processed in the same way as mentioned above.

Character Analysis

The analysis of the characters required by the FETCH routine is
accomplished by the analyzing routines INCHAR, READ, CLAS and
PSYM. READ reads a line of data into memory; INCHAR locates the
non-blank characters; CLAS determines the type of the character; and
PSYM packs a string of letters and digits into an eight character two
word storage area SYM.

In reading a line of data into memory, READ first checks for end
of data and if at end of data, prints the message END OF DATA and terminates *TURM. If not at end of data, READ blanks the storage area of memory allocated for the line. READ then reads a line of data into this storage area and checks for end of file. If it is at end of file, READ returns to the first location following the call, returning End of File. If it is not at end of file, the line counter is incremented, converted to decimal and stored in the storage area LINE. READ will then load the last two words of the storage area and check for a field overflow by shifting the two words one storage word right and filling the first original location with blanks. This spacing will indicate when printed that a field overflow has been encountered.

READ then summons the support routine PRINTB to print the line. Finally the return address is incremented to facilitate a normal return to the calling routine by returning to the second location following the call of READ.

The main function of INCHAR is the location of non-blank characters in the storage area CARD. Thus upon summoning, INCHAR checks the column counter for the end of the work area i.e. column 72. If the column counter is at the end of the work area, the READ routine is summoned to read the next line of data. If the READ routine returns End of File, either the word DONE was not at the end of the quintuple list, or the Exec card did not have the correct format. In either case, the ERRPRINT routine is summoned to print the existing
errors previously encountered. This is then followed by the message
NO DONE CARD; a blank line; and the message ABNORMAL END.
INCHAR will then return to START, the beginning of the MAIN routine,
to try to process another Turing Machine. However, if the READ
routine returns normally, INCHAR loads and checks each character
in the storage area for a non-blank character. If the character is a
blank, it is ignored, the column counter is incremented, the check
for the end of the work area is made and the process continues. When
a non-blank character is found, it is compared to the BCD code for a
dollar sign, ($). The dollar sign represents a blank for the internal
workings of the Turing Machine. Thus each dollar sign is changed to
the BCD blank character and stored in the storage word LCR, or the
last character read. If the character was not a dollar sign, it is
simply stored in LCR. INCHAR will then return to the calling routine.

CLAS determines the type of character being scanned by the
builder. The characters are divided into three main groups: the
first being special characters, i.e. any BCD character which is not
a letter or a digit; the second being a digit, any decimal digit from
zero to nine inclusively; the third being letters from A to Z. The
BCD code representation of the character is then used as an index to
a table called CLTAB. CLTAB is a table of sixteen computer words
with four character locations per word or sixty-four character loca-
tions. Each location corresponds directly with the BCD code
representation of the character. The locations are also coded such that in the character location for a special character is a zero; in the character location for a digit is a one; and in the character location for a letter is a two. Using the BCD code representation of the character as an index to the table, the zero, one or two that is loaded from the table can be added to the return address of CLAS, which will in turn indicate to the calling routine the type of the character being scanned. Thus the special character return from CLAS will be the location in the calling routine directly following the call. In the same sense, the digit return will be to the second location following the call of CLAS, and the letter return will be to the third location following the call of CLAS. The use of CLTAB effectively determines the type of character for the many functions of the FETCH routine.

The PSYM routine packs a letter-digit string left-justified in a two word storage area called SYM. PSYM allows eight characters to be stored in SYM. The characters must be a combination of letters and digits. In most cases, the string will be a specific letter string needed by the calling routine. Any string longer than eight characters will cause PSYM to store the first seven characters of the string and the last character of the string. Thus PSYM fills SYM with blanks and proceeds to load characters from the storage area CARD. Each character is sent to CLAS to determine its type, i.e.
special character, digit or letter. If it is a letter or digit, the character is stored in SYM. When a special character is encountered, the PSYM routine terminates and returns to the calling routine with the character string packed in SYM.

**Secondary Support Routines**

The secondary support routines are PAGE, PRINTB, DECONV and BICONV. These routines supply needed output for the FETCH routine, and the binary-to-decimal decimal-to-binary conversions for the input and output of the quintuples.

PAGE simply causes a page eject and the printing of the Heading at the top of the new page. PAGE also resets the page line counter before returning to the calling routine.

PRINTB is the main output routine for the builder. It prints a twenty-six word storage area LINE, which consists of the line number and the storage area CARD. After printing the line, PRINTB checks the page line counter. If the page line counter is greater than 56, the routine PAGE is summoned to start a new page. If less than 56 or upon returning from PAGE, PRINTB increments the page line counter and returns to the calling routine.

The DECONV routine converts a decimal number to a binary number. DECONV will first process the sign of the number then proceed to convert the number using the method of multiplying the largest
powered tens-digit by ten and adding the next lower powered tens-
digit, multiplying this result by ten and adding the next lower powered
tens-digit, etc., continuing the process until the ones-digit has been
added into the number. Once the digit has been converted to binary,
the sign will convert the number to negative or positive and the con-
verted number returns with the return of DECONV to the calling
routine.

The BICONV routine converts a binary number to decimal and
returns to the calling routine with the converted number in the address
specified by the call of BICONV. The method of conversion consists
of dividing the number by ten and storing the remainder right-justified
in the location specified by the call of BICONV, then dividing the
quotient by ten again, and storing the remainder etc. Once the
quotient becomes zero, the last remainder is stored and the number
has been converted. The sign is stored in the number and BICONV
returns to the calling routine.
THE EXEC ROUTINE

The EXEC routine completes the execution phase of the Turing Machine. EXEC operates on a Turing Machine as specified by the builder, assuming there are no terminating errors. The actual simulation of a Turing Machine is carried out by the EXEC routine.

EXEC first accepts an optional parameter type string indicating: the starting point for the simulation; the output interval for the complete configurations; the starting point for the Information Part; that a Turing Machine writes a second Turing Machine to be sent to the builder for processing and back to EXEC for execution; and finally whether or not there is a specific Information Part to be worked on.

Upon completion of the parameter processing, EXEC will execute the Turing Machine stored in memory by the builder or by the file function if the builder was not used. Thus the EXEC routine is divided into three sections: the processing of the optional parameters; the execution of the Turing Machine as specified by these parameters and as designed by the builder; and the support routines for inputting and outputting of the complete configurations, and the searching of the quintuple list.

Processing the Parameters

The optional parameters are: C, I, Q, S, T, IP, and DONE.
The list is considered optional, since none of the parameters are actually needed, with the exception of one of the last two mentioned. The parameter list has been designed for the User's benefit for the inputing and the starting of the execution of the Turing Machine.

The Q-parameter states the starting quintuple label to be used by the execution. Q must be a decimal number of the same form as any quintuple label. Thus it follows it must be within the range 1 to 1023 inclusively. If the Q-parameter is not used, the starting quintuple label is considered to be one.

The I-parameter specifies the print interval for the complete configurations. Again the I-parameter must be a decimal number. If I is set to three, every third complete configuration will be printed; if set to one, every complete configuration will be printed. If the I-parameter is not used, the print interval will automatically be set to zero and only the final complete configuration will be printed.

The C-parameter specifies the column in the Information Part for the beginning of the processing. The Information Part will be read into memory in the same format as it appears on the line or a card, and thus the C-parameter specifies which column the first symbol being scanned is in. If the C-parameter is not used, the column will automatically be considered to be column one and the
process will begin by loading the character in that position. The C-parameter must be a decimal number within the range of -2000 to 2000 exclusively.

The T-parameter simply notifies the routine that this Turing Machine writes another Turing Machine which also needs to be processed. Thus the T-parameter simply sets a flag that will signal *TURM that a continuation of the program is needed for the processing of the second Turing Machine and that the clean-up will be carried out at a later time.

The S-parameter specifies the area of the Information Tape to be used for the Information Part. The Information Part is stored on the Information Tape as specified by the S-parameter. The Information Tape is itself a storage area of a 1000 words of memory which consists of four characters per word or 4000 possible character positions. The tape is divided into two distinct parts. This allows the Information Part to expand in two directions depending upon the needs of the Turing Machine, Figure 3. Thus the Information Tape is considered to have 2000 positive character locations and 2000 negative character locations with the center being at zero. The C-parameter may specify any of these locations for the starting point of the first quintuple. The S-parameter, on the other hand, will allow the Information Part to begin in any of these locations. This then allows the negative field for the input of the Information Part. The S-parameter
is not used, the Information Part is stored in the Information Tape as it appears on the input card, starting at column one. If used the Information Part will be stored as it appears on the input card but indexed by the S-parameter.

The only exception to the optional parameter list are the IP-parameter and the word DONE. The IP-parameter must be used if there is an Information Part to be inputed. The Information Tape in the technical sense will always contain an Information Part. This so-called Information Part consists of an Information Tape filled in every character location with blanks. However, if the User decides upon a specific Information Part other than the blank tape, the IP-parameter is required. The IP indicates that a specific Information Part follows the Exec card and must be inputed into the Information Tape before the execution phase of the Turing Machine can begin. On the other hand, if the User does not have a specific Information Part, the word DONE must be the last parameter on the Exec card. The word DONE will indicate the end of the parameter list and that a blank tape will be used for the execution phase. EXEC upon encountering the IP-parameter will then read into the Information Tape the specific Information Part to be worked upon. This Information Part will be stored as indexed by the S-parameter, if so used.

Figure 3. The Information Tape.
The Execution

If there is an Information Part to be inputed, the INFOIN routine is summoned to accomplish the task. The C-parameter is then used to load the symbol being scanned and the execution process begins. The symbol being scanned is logically "or"ed into the quintuple label and the SERCH routine is summoned to locate this quintuple address in the quintuple list. If the quintuple is found by SERCH, the location of this quintuple in the storage word STPTR is used to load the replacement conditions. Once the replacement conditions have been loaded and stored in the word RSTAT, RSTAT is compared to the word STOP. If the replacement conditions are the same as the word STOP, a builder error had been encountered and the quintuple could not be computed. Thus EXEC will transfer control to the execution stop section STOP. If it is not the word STOP, the replacement conditions in the word RSTAT are separated into the component parts by storing the move code in the word MC, the replacement label in the area specified for the quintuple label, and the replacement symbol in the Information Tape indexed by the Information Part column counter. The print interval counter checks to see if the print interval has been satisfied. When satisfied, the INFOUT routine is summoned to print the complete configuration. If it is not satisfied, the print interval counter is incremented. The move code in MC is then used to
increment or decrement the Information column counter as required. The stop flag, SFLAG, is checked to determine if a stop condition has been reached. When the replacement label is a zero, the normal stop as determined by the Turing Machine has been reached, and EXEC transfers control to the execution terminating section STOP. If the stop condition has not been reached, EXEC loads the next character from the Information Part and logically "or"s the character into the quintuple label, forming the quintuple address for the next Turing Machine instruction. Again SERCH is summoned and the process continues.

If SERCH does not find the quintuple address in the quintuple list, a flag is set to indicate that the required quintuple is missing, and EXEC again transfers control to the execution stop section, STOP.

The execution stop section, STOP, indicates the execution phase has been completed. Thus the final complete configuration is prepared for output. The last column being scanned by the Information Part column counter is stored in the message COLUMN=. If there was a missing quintuple, the message MISSING QUINTUPLE= is printed with the last quintuple address. If the Turing Machine stopped normally, the message QUINTUPLE= is printed with the last quintuple address. The INFOUT routine is then summoned to print the final complete configuration. Finally the TM flag is checked. The TM flag is set if the T-parameter was used in the parameter
list on the Exec card. If it is set, the Turing Machine just processed has written a second Turing Machine and EXEC transfers to the STM routine in MAIN to set up this second machine for input. If it is not set, the simulation of this Turing Machine has been completed. Therefore EXEC transfers to START in the MAIN routine to re-initialize the tables, counters and pointers for the input of the next Turing Machine to be processed.

If EXEC encounters any errors in the parameters on the Exec card, the message **ILLEGAL CONTROL STATEMENT** is printed and EXEC transfers control to the STOP2 clean-up routine in MAIN.

**Supporting Routines**

During the execution phase of the Turing Machine, the supportive routines INFOIN, INFOUT and SERCH may be used frequently. INFOIN, unlike the other two, is used in only one phase of the execution. INFOIN's main function is to read into memory the Information Part, if there is a specific Information Part to be worked upon. The INFOIN routine in itself is basically the same as the READ routine in FETCH. Thus INFOIN first checks the status of the input unit for a possible end of data. If it is at end of data, the message **END OF DATA** is printed and *TURM terminates. If it is not at end of data, twenty words of the Information Part are read and stored in the storage area CARD. INFOIN then checks for end of file. If it is not
at end of file, the twenty words are printed and INFOIN returns to
the calling routine normally, i.e. to the location directly following
the call. If it is at end of file, the return address becomes the
second location following the calling of INFOIN. EXEC will accept
the twenty words in CARD and store them into the storage area
allocated for the Information Part.

INFOUT, on the other hand, is responsible for the output of the
complete configurations. The printing of the complete configurations
involves a great deal of complexity. Since the Information Tape
consists of 1000 words or 4000 characters for storage, and since the
greater part of this area is filled with blanks, it is not practical to
output the entire storage area for each intermediate complete con-
figuration. To remedy this, only the storage area that contains the
Information Part will be printed. Thus the beginning and ending of
the Information Part must be determined for the output. Also,
since the print line for the Printer is limited to 136 characters or
34 words and the possible Information Part has a limit of 1000
words, there must be some type of capability to output the required
number of lines for the entire Information Part. Finally, for the
User's benefit, an up-arrow, (↑), is used to indicate the symbol
being scanned with the action desired having been carried out. With
this up-arrow will be, at the end of the line, an indication of the
quintuple label used for the action; the original symbol being scanned;
and the column in which the action occurred. Thus INFOUT becomes a complicated routine with several flags, checks and comparisons for the proper output of the complete configurations.

To find the Information Part, INFOUT checks each word of the Information Tape until either the column of the Information Tape being scanned, or the Information Part itself is located. If neither are within the limits of the Information Tape, the task is considered hopeless and INFOUT returns to the calling routine. Once either the column being scanned or the Information Part has been located, INFOUT begins the process of printing the Information Tape. To facilitate the limits of the line printer, only 24 words of the Information Tape will be printed at a time. Five additional words are used to accommodate the quintuple label, symbol being scanned and the column being scanned. And two blank words are used preceding the output line, to eliminate the problems that may occur because of the carriage control. If within the 24 words the column being scanned has been encountered, a flag is set to indicate that an additional line need be printed with the up-arrow to show the symbol being scanned. Thus upon completion of storing the 24 words, the flag indicates that two lines of output are needed. The first line will contain the Information Part section in which the work is presently being done, and the second will be blank with the exception of the up-arrow at the position of the column being scanned. The actual column for the
up-arrow is determined by retaining the complement of the column each time two words of the Information Tape are stored in the output line. Once the 24 words have been printed, the output line is filled with blanks and the column for the up-arrow is determined by subtracting the complement from the total number of possible characters in the line, i.e. 96. The up-arrow is then stored in the appropriate column and the line is again printed. The preceding line will also contain the quintuple label, symbol being scanned and the column of the Information Tape being scanned. The quintuple label and the column will both be converted to decimal before being stored in the output line. All lines preceding and succeeding the line containing the character being worked upon are simply stored in the 24 words and printed. Once the up-arrow is printed, a second flag is set to indicate the checks for the column no longer need be used. At this time also a second check begins for the end of the Information Part on the Information Tape. Three words of blanks or 12 consecutive blanks are assumed to be enough evidence to indicate the end of the Information Part. The 12 blanks are not carried over from one 24 word section to the next. This check is not made until after the 24 word line containing the column being scanned has been printed. Upon reaching the line with the three words of blanks, INFOUT prints the line, resets the print interval counter and returns to the calling routine.
The SERCH routine does not contain the complexity of the INFOUT routine, but does make use of several pointers. For SERCH there is a top of the list pointer, bottom of the list pointer, a starting pointer, an up pointer and a down pointer. The starting pointer, STPTR, will indicate the beginning of the search of the quintuple list. It will be the location of the last quintuple found in the quintuple list. The up pointer will be incremented searching up the table from the starting pointer, and the down pointer will be decremented searching down the table from the starting pointer. The SERCH routine works on the premise that most of the quintuples in the quintuple list will be within close proximity of the last quintuple referenced. Working on this premise, SERCH will simultaneously search up and down the list from the starting pointer. SERCH checks the quintuple at the location specified by the starting pointer, if the quintuple is not found, SERCH checks the quintuple at the location specified by the up pointer. If the quintuple still has not been found, the up pointer is incremented and the up pointer is then compared to the down pointer for equality. If they are equal, the entire list, with the exception of the present quintuple, has been checked and the quintuple has not been found. If they are not equal, the up pointer is compared to the top of the list pointer for equality. If they are equal, the bottom of the list pointer is stored in the up pointer to allow the process to continue. If they are not equal, the quintuple at the
location of the down pointer is checked. If it is not the desired quintuple, the down pointer is decremented then compared again to the up pointer for equality. If they are equal, the entire list has been checked with the exception of the present quintuple. If they are not equal, the down pointer is compared to the bottom of the list pointer for equality. If they are equal, the top of the list pointer is stored in the down pointer. If they are unequal, the process continues by checking the quintuple at the location specified by the up pointer. This cyclic searching continues until either the quintuple is found or until the up pointer is equivalent to the down pointer. In the case of the equality of the pointers, the last quintuple is checked. If it is not the desired quintuple, SERCH returns to the calling routine as not found. If the quintuple is found in the list, the address of the quintuple is stored in STPTR. Thus STPTR will give the address of the desired quintuple to the calling routine and will also be the starting point for the next call of SERCH. SERCH then returns to the calling routine by incrementing the return address and returning to the second location following the call, i.e. SERCH returns found.

The main requirements for the simulation of the Turing Machine have now been satisfied. The discussed routines and their supportive routines will process the Turing Machines within the limits stated.
THE FILES ROUTINE

The basic requirements for the Turing Machine simulation have now been completed. The only additional routine to be discussed is the FILES routine. Although not an essential part of the simulation, the FILES routine has been designed for the convenience of the User. FILES main functions consist of the saving, getting, correcting or the destroying of any Turing Machine. The essential reason for the existence of the FILES routine is the elimination of the tedious work *TURM requires to build a Turing Machine every time the use of the machine is demanded. Once a Turing Machine has been saved, it can be quickly corrected, run and/or destroyed on Teletype or Batch. Thus FILES was constructed as a time saving device for the User.

With the exception of the save and destroy functions, the FILES routine replaces the builder in preparing a Turing Machine for the execution phase. Once the builder has completed its work, FILES can save the constructed Turing Machine on a file, then summon the machine when needed, or destroy the machine when the User is finished with it.

FILES in itself is divided into the four major routines SVEF,
CRTCF, GF and DESF, and the two minor but supporting routines RWSRCH and DIRECT. There is also a routine called EDITS, which is utilized exclusively by CRTCF and will be discussed with CRTCF.

The Four Major Functions

Upon entry into the FILES routine, the desired file function in the storage word FSYM is sent to the RWSRCH routine to determine if the function is a legal function, i.e. SAVE, GET, CORRECT, or DESTROY, and if so to determine the index to the jump table JTAB, which will transfer control to the desired routine dictated by the function. If the function is not a legal function, FILES will automatically transfer control to the clean-up routine in MAIN, STOP2. The jump table, JTAB, is a table of four jump instructions which will transfer control to the routine handling the desired file function.

The Save routine, SVEF, will be summoned immediately after the building process has been completed. The main function of the SVEF routine is the saving of a Turing Machine on a file under the name desired by the User. The Turing Machine name has been previously stored in the storage word SYM. Thus, SVEF calls the DIRECT routine to determine if a Turing Machine by this name has previously been saved. If the name is in the list of Turing Machine names and so discovered by DIRECT, the message FILE ALREADY EXISTS BY THIS NAME is printed and the logical unit used by the
DIRECT routine is unequipped to allow for the execution of another Turing Machine. SVEF then returns to the routine calling FILES. If the Turing Machine name was not found by the DIRECT routine, it will be necessary to store the name in the directory, i.e. the list of saved Turing Machine names. SVEF then searches the directory linearly for a possible blank area. The blank area would exist only if at some previous time a Turing Machine had been destroyed. If such a blank area is discovered, the Turing Machine name will be stored in that area. If such an area is not discovered, the Turing Machine name will be stored at the end of the existing list of names. This special search eliminates the possible waste of the storage area within the directory. The storage location of the Turing Machine name in the directory becomes an index for an internal table ITAB. ITAB consists of a list of names to be used by the Systems directory. The purpose of this table revolves around the possibility of conflicting names used for the Turing Machines and those used by the development of other programs. To eliminate this possibility, the name table ITAB consists of 60 names which will correspond directly to the name of the Turing Machine to be saved. In actuality, the Turing Machine name is saved in a file called DIRTJ and its location in this file is the index to the name in ITAB under which the Turing Machine is saved by the System. Once the corresponding name in ITAB has been computed, SVEF will equip logical unit 1 to a
scratch file, write the Turing Machine onto this file, follow it with a file mark and rewind the file. SVEF then loads the corresponding name and saves the file under that name. Both logical unit 1 and logical unit 3 are then unequipped for they are no longer needed, and SVEF returns to the routine calling FILES. The Turing Machine has now been saved under the name desired and upon User request will be loaded into memory.

The Correct routine, CRTCF, is designed to eliminate the errors encountered by the builder. There are only two types of errors that are correctable. The first are those errors encountered by a missing quintuple or an erroneous quintuple address. The second type are those errors encountered by an erroneous replacement field. In the first case, the quintuple has not been stored in the list and therefore must be added to the list. In the second case, the erroneous quintuple is stored in the quintuple list and must be changed. Both tasks are accomplished by the EDITS routine when summoned by CRTCF. CRTCF is then responsible for setting up the quintuple list for the EDITS routine and saving the corrected Turing Machine.

CRTCF first summons DIRECT to locate the Turing Machine in question. If DIRECT does not locate the Turing Machine name, the message **FILE NAME NOT IN DIRECTORY** is printed and CRTCF transfers control to the clean-up routine in MAIN, STOP2. If the
The Turing Machine name is located, the DIRECT routine returns to CRTCF with the index to the corresponding systems name in ITAB. The logical unit 1 is then equipped to the Systems name and the Turing Machine is read into memory. The EDITS routine is summoned to make the needed corrections. Upon return from EDITS, the corresponding systems name is deleted from the Systems directory and logical unit 1 is unequipped for it is no longer needed. A scratch file is then equipped to logical unit 2 and the corrected Turing Machine is written onto this file. The file is then followed by a file mark and rewound. Finally the file is saved under the corresponding systems name and the logical unit 2 and logical unit 3 are unequipped. CRTCF then returns to the routine calling FILES.

The EDITS routine when summoned by CRTCF will add or replace any erroneous quintuples. To indicate the beginning of the process, the list of quintuples to be altered must be preceded by the word **ADD** or the word **REPLACE**. The list must also be concluded by the word **DONE**. Finally the quintuples must be of the same format as the original quintuples used by the builder. In this way, EDITS can make use of the analyzing sections of FETCH. Therefore a check is made for the word **ADD** or **REPLACE** by summoning PSYM to pack the letter string and by summoning RWSRCH to search for the appropriate word in the reserve word table. If it is not found by RWSRCH, EDITS unequips logical unit 1 and logical unit 3 and transfers control
to the MAIN clean-up section STOP2. On the other hand if it is found, EDITS after setting the edit return flag, summons FETCH3 to set up the quintuple address. Upon returning from FETCH3, the done flag is checked to see if the entire correction list has been computed. If it is not set, SERCH is summoned to check the quintuple list for the constructed quintuple address. If SERCH locates the quintuple, FETCH2 is summoned to construct the replacement conditions and stores these conditions in the location specified by SERCH. If SERCH does not locate the quintuple, FETCH2 is summoned to construct the replacement conditions storing these conditions at the bottom of the existing quintuple list. In either case after storing the quintuple, EDITS again calls FETCH3 to process the next quintuple. When the done flag has been set, the corrections have been made and EDITS returns to the calling location in CRTCF.

Once the Turing Machine has been saved and/or corrected, it may be summoned at any time for the execution of data. The Get routine, GF, is simply designed to locate the Turing Machine in question and store that machine in memory. Thus GF summons DIRECT to locate the Turing Machine. Again if DIRECT cannot locate the machine, the message FILE NAME NOT IN DIRECTORY is printed and GF transfers to the MAIN clean-up routine STOP2. If it is located, GF equips logical unit 1 to the corresponding systems name and proceeds to read the Turing Machine into memory. Once
the machine is in memory, logical unit 1 is unequipped and GF returns to the routine calling FILES.

When destroying a Turing Machine, the only requirements needed are deletion of the corresponding systems name and the deletion of the Turing Machine name. Thus DIRECT is summoned to locate the name in the directory. If DIRECT cannot locate the name, DESF simply transfers to the MAIN clean-up routine STOP2. If the name is located, DESF deletes the corresponding systems name from the Systems directory and unequips logical unit 1, the unit used. Then DESF determines the location of the Turing Machine name and replaces the name with blanks. Logical unit 3 is then rewound and the modified list of Turing Machine names is written on the file. Finally logical unit 3 is rewound again and unequipped. DESF then transfers to the MAIN clean-up routine STOP2.

Supporting Routines

The supporting routines used by FILES are RWSRCH which searches a specific table of reserve words containing the file functions in question, and DIRECT which reads into memory a list of Turing Machine names and searches this list for a specific name.

RWSRCH searches a table, RWTAB, for a file function. RWSRCH uses a linear search to locate the function desired. The first four functions refer to the routines designed to process the
file, i.e. SAVE, CORRECT, GET and DESTROY. The last two functions are equivalent and are used by the EDITS routine, i.e. REPLACE and ADD. If the function is among the first four, RWSRCH processes the jump table index used in the calling routine by using the position in the table as the index. If the function is one of the last two, RWSRCH simply returns to the calling routine. In either case, RWSRCH will return to the calling routine as found. However if the word in question is not in the table, RWSRCH prints the message **ILLEGAL FILE CALL STATEMENT** and returns to the calling routine as not found.

The DIRECT routine on the other hand must first read a file, DIRTJ, into memory. Therefore DIRTJ is equipped to logical unit 3 and read into memory to be searched. Once DIRTJ is in memory, DIRECT searches the name list linearly for the name in question. If the name cannot be found in the list, DIRECT simply returns to the second location following the call, thus returning not found. If the name is found, the location of the name becomes the index for the corresponding systems name. The name list in DIRTJ allows two words per Turing Machine name. However the corresponding systems name is only one word in length. The index is therefore divided by two and stored as the index for the corresponding systems name. DIRECT then returns to the location directly following the call, thus returning found.
EPILOGUE

This completes the organization and operations of *TURM, the Turing Automaton Simulator Program. The use of *TURM may be beneficial to all Turing Machine designers. *TURM has not been designed to write the machines, but rather to execute them, saving the tedious hours of checking numerous possibilities for each Turing Machine. *TURM is a time saver and a work saver. It is limited to the confines of the computer, but even though limited it may save the User hours of frustration. *TURM is far from being the perfect simulator, but it does perform the simulation well and quickly to human standards.
APPENDICES
APPENDIX I

TEST CASES

Test 1

Test 1 exemplifies the use of a simple Turing Machine called RECADD, a recursive binary adder. RECADD requires as data a string of binary numbers separated by plus signs and delimited by H's. The test shows the quintuple list, the Save file function, and the Exec card followed by the Information Part as each appears on the card, with the exception of the line number which is printed by *TURM. The printed results show the intermediate complete configurations as determined by the I-parameter; the terminating column; the terminating quintuple, which indicates the Turing Machine worked as designed; and the final complete configuration, giving the answer derived by the machine.

<table>
<thead>
<tr>
<th>TURM</th>
<th>RECADD</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1 H 13 H 1</td>
</tr>
<tr>
<td>0002</td>
<td>2 0 2 0 1</td>
</tr>
<tr>
<td>0003</td>
<td>3 B 3 B 2</td>
</tr>
<tr>
<td>0004</td>
<td>5 H 5 B 2</td>
</tr>
<tr>
<td>0005</td>
<td>5 B 5 B 2</td>
</tr>
<tr>
<td>0006</td>
<td>6 0 6 0 1</td>
</tr>
<tr>
<td>0007</td>
<td>7 1 7 1 2</td>
</tr>
<tr>
<td>0008</td>
<td>8 A 8 A 2</td>
</tr>
<tr>
<td>0009</td>
<td>9 0 6 1 1</td>
</tr>
<tr>
<td>0010</td>
<td>10 B 10 0 2</td>
</tr>
<tr>
<td>0011</td>
<td>11 0 11 0 1</td>
</tr>
</tbody>
</table>
H10101********BH11H  +0011 B +0019
H10101*********1BH    +0007 * +0019
H1011BH*********1BH    +0006 B +0011
H1011BH*********1BH    +0006 * +0021
H1011BH*********BBH    +0007 * +0017
H110BB*********BBH     +0006 0 +0009
H110BB*********BBH     +0006 * +0019
H110BB*********BBH     +0003 * +0019
H11000*********BBH     +0010 0 +0009
H11000*********BBH     +0011 * +0013
H11000**********H      +0011 B +0023
H11000*****           +0012 * +0017

COLUMN = +0012
QUINTUPLE = +0000 0
H11000H               +0000 0 +0012
Test 2

Test 2 exemplifies the use of a simple Turing Machine called MULTPLY. MULTPLY is a binary multiplier requiring a data string of two binary numbers separated by a (*), and a starting column for the quintuple address. The test shows the quintuple list, the Save function, the Exec card and the Information Part as they appear on the card. The printed results show the intermediate complete configurations, again as determined by the I-parameter; the terminating column; the terminating quintuple, which is in fact missing from the quintuple list; and finally the terminating Information Part, with the work thus far accomplished.

<table>
<thead>
<tr>
<th>TURM</th>
<th>MULTPLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1 1 1 1 2</td>
</tr>
<tr>
<td>0002</td>
<td>2 + 3 + 1</td>
</tr>
<tr>
<td>0003</td>
<td>4 1 5 $ 1</td>
</tr>
<tr>
<td>0004</td>
<td>5 * 6 * 1</td>
</tr>
<tr>
<td>0005</td>
<td>6 A 7 A 2</td>
</tr>
<tr>
<td>0006</td>
<td>7 * 13 * 1</td>
</tr>
<tr>
<td>0007</td>
<td>8 * 8 * 2</td>
</tr>
<tr>
<td>0008</td>
<td>9 $ 9 $ 2</td>
</tr>
<tr>
<td>0009</td>
<td>10 1 12 B 1</td>
</tr>
<tr>
<td>0010</td>
<td>10 C 10 C 2</td>
</tr>
<tr>
<td>0011</td>
<td>11 A 11 A 2</td>
</tr>
<tr>
<td>0012</td>
<td>12 0 12 0 1</td>
</tr>
<tr>
<td>0013</td>
<td>12 C 12 C 1</td>
</tr>
<tr>
<td>0014</td>
<td>13 0 13 0 2</td>
</tr>
<tr>
<td>0015</td>
<td>13 B 13 1 1</td>
</tr>
<tr>
<td>0016</td>
<td>14 B 14 1 2</td>
</tr>
<tr>
<td>0017</td>
<td>15 B 15 O 2</td>
</tr>
<tr>
<td>0018</td>
<td>16 $ 16 $ 1</td>
</tr>
<tr>
<td>0019</td>
<td>17 0 17 0 1</td>
</tr>
</tbody>
</table>
0020 18 $ 18 $ 2 18 * 18 * 2 18 + 3 + 2 19 $ 19 $ 1
0021 19 * 20 $ 1 20 1 20 $ 1 20 0 20 $ 1 20 $ STOP $ 0
0022 3 * 4 * 2
0023 /SAVE  MULTIPLY
0024 EXEC  C=7  Q=1  I=10  IP
11101*11

\[
\begin{align*}
0 + 11101*11 & \quad +0003 \quad * \quad +0012 \\
0 + 1110 * 1B & \quad +0009 \quad 0 \quad +0010 \\
B + 1110 * 1B & \quad +0012 \quad 0 \quad +0010 \\
B + 1110 * BB & \quad +0009 \quad 1 \quad +0008 \\
BB + 1110 * BB & \quad +0012 \quad 0 \quad +0010 \\
BB + 1110 * 11 & \quad +0013 \quad * \quad +0012 \\
11 + 1110 * 11 & \quad +0016 \quad 1 \quad +0004 \\
11 + 1110 * 11 & \quad +0017 \quad 1 \quad +0014 \\
11 + 1110 * 110 & \quad +0018 \quad + \quad +0006 \\
\end{align*}
\]

COLUMN = +0006
MISSING QUINTUPLE = +0003 +
11 + 1110 * 110 +0003 * +0006
Test 3

Test 3 is the resubmitting of the Turing Machine used by Test 2. The missing quintuple in Test 2 was the result of an erroneous quintuple in the design of the machine. Thus Test 3 shows the Correct function followed by the Exec card and the Information Part. The intermediate complete configurations, the column, the terminating quintuple and the final complete configuration indicate the Turing Machine worked as designed.

/CORRECT MULTIPLY

0001 REPLACE 18 + 3 + 1 DONE
0002 EXEC C=7 Q=1 I=10 IP
1101*11

TURM MULTIPLY

\begin{align*}
0+11101*11 & \quad +0003 \ast +0012 \\
0+1110 \ast 1B & \quad +0009 \ 0 +0010 \\
B+1110 \ast 1B & \quad +0012 \ 0 +0010 \\
B+1110 \ast BB & \quad +0009 \ 1 +0008 \\
BB+1110 \ast BB & \quad +0012 \ 0 +0010 \\
BB+1110 \ast 11 & \quad +0013 \ast +0012 \\
11+1110 \ast 11 & \quad +0016 \ 1 \ast 0004 \\
11+1110 \ast 11 & \quad +0017 \ 1 +0014 \\
11+1110 \ast 110 & \quad +0018 \ + +0006
\end{align*}
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>11BB+1</td>
<td>*110AA</td>
<td>+0012</td>
<td>+ 0006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11BB+1</td>
<td>*110AA</td>
<td>+0006</td>
<td>A +0016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11BB+1</td>
<td>*11AA</td>
<td>+0008</td>
<td>+ 0006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1BBB+1</td>
<td>*11AAA</td>
<td>+0012</td>
<td>+0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1BBB+1</td>
<td>*1BAAA</td>
<td>+0009</td>
<td>+0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBBB+1</td>
<td>*1BAAA</td>
<td>+0012</td>
<td>B +0004</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBBB+1</td>
<td>*1BAAA</td>
<td>+0006</td>
<td>B +0014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBBB+1</td>
<td>*BBAAA</td>
<td>+0011</td>
<td>B +0004</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCBBB+1</td>
<td>*BBAAA</td>
<td>+0012</td>
<td>+0008</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCBBB+1</td>
<td>*1100A</td>
<td>+0013</td>
<td>A +0016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BCBBB+1</td>
<td>*11000</td>
<td>+0013</td>
<td>+0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100111+1</td>
<td>*11000</td>
<td>+0015</td>
<td>+0000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100111+1</td>
<td>*11000</td>
<td>+0016</td>
<td>+0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100111+1</td>
<td>*110000</td>
<td>+0018</td>
<td>0 +0016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100111+1</td>
<td>*110000</td>
<td>+0018</td>
<td>+ 0006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100111+</td>
<td>*110000</td>
<td>+0006</td>
<td>1 +0014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100111+</td>
<td>*11000A</td>
<td>+0008</td>
<td>1 +0014</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10011B+</td>
<td>*11000A</td>
<td>+0012</td>
<td>+ 0006</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10011B+</td>
<td>*11000A</td>
<td>+0006</td>
<td>0 +0016</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10011B+</td>
<td>*1100AA</td>
<td>+0008</td>
<td>+0010</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001BB+</td>
<td>*1100AA</td>
<td>+0012</td>
<td>+0008</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001BB+</td>
<td>*110AAA</td>
<td>+0007</td>
<td>0 +0016</td>
</tr>
</tbody>
</table>
\[
\begin{align*}
1001BB & \quad *110AAA & +0008 & + & +0006 \\
100BB & \quad *110AAA & +0012 & + & +0010 \\
100BBB & \quad *11AAAA & +0008 & * & +0012 \\
10ABB & \quad *11AAAA & +0010 & 0 & +0002 \\
10ABB & \quad *11AAAA & +0012 & * & +0012 \\
10ABB & \quad *1BAAAA & +0009 & + & +0008 \\
1BABB & \quad *1BAAAA & +0012 & B & +0004 \\
1BABB & \quad *1BAAAA & +0006 & B & +0014 \\
1BABB & \quad *BBAAAA & +0011 & B & +0004 \\
CBB & \quad *BBAAAA & +0012 & + & +0006 \\
CBB & \quad *11AAAA & +0013 & B & +0014 \\
CBB & \quad *110000 & +0013 & 1 & +0014 \\
CBB & \quad *110000 & +0014 & B & +0004 \\
101011 & \quad *110000 & +0016 & 1 & +0004 \\
101011 & \quad *110000 & +0017 & 1 & +0014 \\
101011 & \quad *1100000 & +0018 & 1 & +0014 \\
101011 & \quad *1100000 & +0004 & + & +0006 \\
000 & +0020 & 0 & + & +0016 \\
\end{align*}
\]

COLUMN = +0020
QUINTUPLE = +0000  0
1010111 +0000  0  +0020
Test 4 exemplifies the use of a simple Turing Machine called ADDER, a binary adder. It shows the output of the intermediate complete configurations and the terminating conditions. The test also exemplifies the retrieval of the adder for a second time, allowing the machine to work upon a second set of data printing the results.

/GET ADDER
0001 EXEC C=6 Q=1 I=1 IP
110+11111

TURM ADDER
110+11111 +0001 + +0006
↑ +0002 1 +0007
110+11111
↑ +0002 0 +0008
110+11111
↑ +0002 + +0009
11++11111 +0003 0 +0008
↑ +0004 + +0009
11++11111
↑ +0004 1 +0010
11++11111
↑ +0004 1 +0011
11++11111
↑ +0004 1 +0012
11++11111
↑ +0004 1 +0013
11++11111
↑ +0004 1 +0014
11++11111
↑ +0004 +0015
↑
10010B
  ↑
100101
  ↑
100101
  ↑

COLUMN = +0015
QUINTUPLE = +0000 0
100101

0001  EXEC  C=8  Q=1  I=5  IP
10011+111011

TURM  ADDER

1011+111011
  ↑
1001++111011
  ↑
1001++111011
  ↑
1001++11110A
  ↑
100+++11110A
  ↑
100+++11110A
  ↑
100+++1111BA
  ↑
100+++1111BA
  ↑
10+++1111BA
  ↑
10+++111BBA
  ↑
1+++111BBA
  ↑
1+++11BBBA
++++++11BBBA
↑
++++++11BBBA
↑
++++++0ABBBA
↑
++++++10ABBBA
↑

++10ABBBA
↑
100BBBA
↑
1001110

COLUMN = +0020
QUINTUPLE = +0000 0
1001110
↑

+0003 1 +0008
+0005 + +0013
+0009 1 +0014
+0003 + +0009
+0010 + +0010
+0010 A +0015
+0010 +0020 +0000 0 +0020
Test 5 exemplifies the results of a Turing Machine containing several errors, printing the messages indicating the errors.

<table>
<thead>
<tr>
<th>TURM</th>
<th>ERROR 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1 1 2 1 1</td>
</tr>
<tr>
<td>0002</td>
<td>2 1 2 1 1</td>
</tr>
<tr>
<td>0003</td>
<td>2 0 2 0 1</td>
</tr>
<tr>
<td>0004</td>
<td>2 + 3 + 2</td>
</tr>
<tr>
<td>0005</td>
<td>3 1 5 + 1</td>
</tr>
<tr>
<td>0006</td>
<td>3 0 4 + 1</td>
</tr>
<tr>
<td>0007</td>
<td>3 $ 12 $ 0</td>
</tr>
<tr>
<td>0008</td>
<td>3 + 3 + 2</td>
</tr>
<tr>
<td>0009</td>
<td>4 1 4 1 1</td>
</tr>
<tr>
<td>0010</td>
<td>4 0 4 0 1</td>
</tr>
<tr>
<td>0011</td>
<td>4 $ 6 $ 2</td>
</tr>
<tr>
<td>0012</td>
<td>4 + 4 + 1</td>
</tr>
<tr>
<td>0013</td>
<td>4 A 6 A 2</td>
</tr>
<tr>
<td>0014</td>
<td>4 B 6 B 2</td>
</tr>
<tr>
<td>0015</td>
<td>5 1 5 1 1</td>
</tr>
<tr>
<td>0016</td>
<td>5 $ 7 $ 2</td>
</tr>
<tr>
<td>0017</td>
<td>5 + 5 + 1</td>
</tr>
<tr>
<td>0018</td>
<td>5 A 7 A 2</td>
</tr>
<tr>
<td>0019</td>
<td>5 B 7 B 2</td>
</tr>
<tr>
<td>0020</td>
<td>5 0 5 0 1</td>
</tr>
<tr>
<td>0021</td>
<td>5 1 5 1 1</td>
</tr>
<tr>
<td>0022</td>
<td>6 1 8 B 2</td>
</tr>
<tr>
<td>0023</td>
<td>6 0 8 A 2</td>
</tr>
<tr>
<td>0024</td>
<td>7 1 9 A 2</td>
</tr>
<tr>
<td>0025</td>
<td>7 0 8 B 2</td>
</tr>
<tr>
<td>0026</td>
<td>7 + 8 B 2</td>
</tr>
<tr>
<td>0027</td>
<td>8 1 8 1 2</td>
</tr>
<tr>
<td>0028</td>
<td>A 0 8 0 2</td>
</tr>
<tr>
<td>0029</td>
<td>8 + 3 + 2</td>
</tr>
<tr>
<td>0030</td>
<td>8 A A A 2</td>
</tr>
<tr>
<td>0031</td>
<td>8 B 8 B 2</td>
</tr>
<tr>
<td>0032</td>
<td>9 1 9 0 2</td>
</tr>
<tr>
<td>0033</td>
<td>9 0 8 1 3</td>
</tr>
<tr>
<td>0034</td>
<td>9 + 8 1 2</td>
</tr>
<tr>
<td>0035</td>
<td>10 1 10 1 1</td>
</tr>
<tr>
<td>0036</td>
<td>10 0 10 0 1</td>
</tr>
</tbody>
</table>
0037  10 B 10 1 1
0038  10 $ STOP $ 0
0039  10 + 10 $ 1
0040  10 A 10 0 1
0041  10 + 10 $ 1
0042  DONE

TURM       ERROR1
LINE 0021  MULTIPLE DEFINED QUINT 0005  0001
LINE 0028  ILLEGAL QUINT LABEL A 2081
LINE 0030  ILLEGAL REPLACE, LABEL 0008  000A
LINE 0041  MULTIPLE DEFINED QUINT 0010  000+

0043  /SAVE ERROR1
0044  EXEC  C=6  Q=1  IP
       101+11

COLUMN = +0009

MISSING QUINTUPLE = +0005 +
       10++11
    ↑
Test 6 exemplifies a Turing Machine, SECTM, writing a second Turing Machine called JO. The intermediate complete configurations shows the actual building of JO. The final complete configuration is the second Turing Machine which is submitted to *TURM, saved, and executed by working upon the specific Information Part. JO is a simple Turing Machine which will indicate a change of letters in a letter string, by replacing the changing letter with a B. The final complete configuration indicates that both Turing Machines worked as designed.

<table>
<thead>
<tr>
<th>TURM</th>
<th>SECTM</th>
</tr>
</thead>
<tbody>
<tr>
<td>0001</td>
<td>1 $ 2 J 1 2 $ 3 0 1 3 $ 4 $ 1 4 $ 5 1 1 5 $ 6 $ 1</td>
</tr>
<tr>
<td>0002</td>
<td>6 $ 7 . 1 7 $ 8 $ 1 8 $ 9 2 1 9 $ 10 $ 1 10 $ 11 . 1</td>
</tr>
<tr>
<td>0003</td>
<td>11 $ 12 $ 1 12 $ 13 1 1 13 $ 14 $ 1 14 $ 15 2 1 15 $ 15 $ 1</td>
</tr>
<tr>
<td>0004</td>
<td>16 $ 17 A 1 17 $ 18 $ 1 18 $ 19 2 1 19 $ 20 $ 1 20 $ 21 A 1</td>
</tr>
<tr>
<td>0005</td>
<td>21 $ 22 $ 1 22 $ 23 1 1 23 $ 24 $ 1 24 $ 26 2 1 26 $ 27 $ 1</td>
</tr>
<tr>
<td>0006</td>
<td>27 $ 28 B 1 28 $ 30 $ 1 30 $ 31 3 1 31 $ 32 $ 1 32 $ 33 B 1</td>
</tr>
<tr>
<td>0007</td>
<td>33 $ 34 $ 1 34 $ 35 1 1 35 $ 36 $ 1 36 $ 37 2 1 37 $ 38 $ 1</td>
</tr>
<tr>
<td>0008</td>
<td>38 $ 39 . 1 39 $ 40 $ 1 40 $ 41 S 1 41 $ 42 T 1 42 $ 43 0 1</td>
</tr>
<tr>
<td>0009</td>
<td>43 $ 44 P 1 44 $ 45 $ 1 45 $ 46 . 1 46 $ 47 $ 1 47 $ 48 0 1</td>
</tr>
<tr>
<td>0010</td>
<td>48 $ 49 $ 1 49 $ 50 3 1 50 $ 51 $ 1 51 $ 52 A 1 52 $ 54 $ 1</td>
</tr>
<tr>
<td>0011</td>
<td>54 $ 55 2 1 55 $ 56 1 1 56 $ 57 B 1 57 $ 58 $ 1 58 $ 59 1 1</td>
</tr>
<tr>
<td>0012</td>
<td>59 $ 60 $ 1 60 $ 61 3 1 61 $ 62 $ 1 62 $ 63 B 1 63 $ 64 $ 1</td>
</tr>
<tr>
<td>0013</td>
<td>64 $ 65 3 1 65 $ 66 $ 1 66 $ 67 A 1 67 $ 68 $ 1 68 $ 69 1 1</td>
</tr>
<tr>
<td>0014</td>
<td>69 $ 70 $ 1 70 $ 71 3 1 71 $ 72 $ 1 72 $ 73 . 1 73 $ 74 $ 1</td>
</tr>
<tr>
<td>0015</td>
<td>74 $ 75 S 1 75 $ 76 T 1 76 $ 77 0 1 77 $ 78 P 1 78 $ 79 $ 1</td>
</tr>
<tr>
<td>0016</td>
<td>79 $ 80 . 1 80 $ 81 $ 1 81 $ 82 0 1 82 $ 83 $ 1 83 $ 84 D 1</td>
</tr>
<tr>
<td>0017</td>
<td>84 $ 85 0 1 85 $ 86 N 1 86 $ 87 E 1 87 $ STOP $ 0 DONE</td>
</tr>
<tr>
<td>0018</td>
<td>EXEC C=5 Q=1 I=5 T DONE</td>
</tr>
</tbody>
</table>
TURM JO
0001 $ STOP $ 0 DONE
0002 /SAVE JO
0003 EXEC C=4 Q=1 I=1 IP
AAAABBBAAABBBBBBB

AAAABBBAAABBBBBBB +0001 +0004
↑
AAAABBBAAABBBBBBB +0002 A +0005
↑
AAAABBBAAABBBBBBB +0002 A +0006
↑
AAAABBBAAABBBBBBB +0002 A +0007
↑
AAAABBBAAABBBBBBB +0002 A +0008
↑
AAAABBBAAABBBBBBB +0002 B +0009
↑
AAAABBBAAABBBBBBB +0003 B +0010
↑
AAAABBBAAABBBBBBB +0003 B +0011
↑
AAAABBBAAABBBBBBB +0003 A +0012
↑
AAAABBBAAABBBBBBB +0002 B +0013
↑
AAAABBBAAABBBBBBB +0002 A +0014
↑
AAAABBBAAABBBBBBB +0002 B +0015
↑
AAAABBBAAABBBBBBB +0003 B +0016
↑
AAAABBBAAABBBBBBB +0003 A +0017
↑
AAAABBBAAABBBBBBB +0002 A +0018
↑
AAAABBBAAABBBBBBB +0002 A +0019
↑
AAAABBBAAABBBBBBB +0002 B +0020
↑
AAAABBBAAABBBBBBB +0003 B +0021
↑
AAAABBBAAABBBBBBB +0003 B +0022
↑
COLUMN = +0023
QUINTUPLE = +0000  0

AAAAABAAAAABABAAAAA

+0003  +0023

+0000  0  +0023
Test 7 exemplifies the retrieval of a saved Turing Machine on Teletype using a negative field for the information Part and the starting column.

```
/GET   ADDER
EXEC  C=-90  S=-90  IP
0001   EXEC  C=-90  S=-90  IP

11+1

11 +1

TURM   ADDER
02/20/71  1340

COLUMN = -0087
QUINTUPLE = +0000  0
            100
            ↑  +0000  0  -0087
```
USING THE TURING MACHINE SIMULATOR

*TURM is primarily designed for Batch use. However Teletype can be used effectively and in some cases more efficiently than Batch. The Turing Machine simulator will allow the User to submit any Turing Machine for testing. It will allow the User to save, get, correct, and/or destroy any Turing Machine. It will print any interval of the complete configurations or only the terminating complete configuration as desired. It will work on an Information Part starting at any specified quintuple and any specified column as desired by the User, always printing the terminating quintuple address, terminating column and final complete configuration. It will then proceed to work upon another Turing Machine in the same fashion. *TURM is designed to accept several Turing Machines consecutively, each being handled in the same way.

In order to use the simulator, the User should keep in mind the following definitions:

Turing Machine: A Turing Machine is a logical machine consisting of an alphabet, a tape, a set of machine configurations, and a set of rules using the components.

The Alphabet: The alphabet consists of the following BCD characters:
States: The machine states consist of any decimal number within the range 1 to 1023 inclusively.

Moves: Zero represents no move; One represents a move right one cell; and any other BCD character represents a move left one cell.

Quintuples: A machine configuration consisting of five elements: $Q_i S_i Q'_i S'_i M$, where $Q_i$ represents the present state, $S_i$ represents the present symbol being scanned, $Q'_i$ represents the replacement state, $S'_i$ represents the replacement symbol, and $M$ represents the move.

Quintuple Label: The quintuple label is the present state $Q_i$.

Quintuple Address: The quintuple address is the present state with the symbol being scanned $Q_i S_i$.

Replacement Symbol: The replacement symbol is the symbol that replaces the symbol being scanned $S'_i$.

Replacement Label: The replacement label if the new or next quintuple label to be used upon the completion of the move $Q'_i$. 
Information Tape: The Information Tape is a tape of 4000 cells in which each cell is filled with blanks.

Information Part: The Information Part is the data stored into the Information Tape by the User, limited to 4000 characters.

Quintuple List: The quintuple list is the list of quintuples used to work upon the Information Part as designed.

Title: The title is the name given to any particular Turing Machine by the User.

Batch

Any Turing Machine to be used on the simulator must first call *TURM. This initiates the simulator. For the first run of the simulator, the Turing Machine must have a title card, a quintuple list, an Exec card, an Information Part, if desired, and an end of file card, the last Turing Machine being followed by a Logoff card.

The title of any Turing Machine may consist of any combination of letters or digits from one to eight characters in length. However, the first character of the title must be a letter. The title may appear anywhere on the card between column one and column 72. Nothing else may be on the card. The only exception to this rule will be discussed under the section concerning the Turing Machine writing and executing a second Turing Machine.

The quintuple list is simply a list of quintuples of the form
\( Q_i, S_i, Q'_i, S'_i, M \), where \( Q_i \) is the quintuple label; \( S_i \) is the symbol being scanned; \( Q'_i \) is the replacement label; \( S'_i \) is the replacement symbol; and \( M \) is the move code as defined. Each of the quintuple elements must be separated by at least one blank, and each of the quintuples must be separated by at least one blank. The only exception to this format is the stop quintuple represented by the word \textbf{STOP}. In this case the \( Q'_i \) element must be the word \textbf{STOP}. Examples: 10 A 2 B 1, 12 D STOP 10 0 DONE

Since the order of the quintuple is the only requirement for *TURM, the number of quintuples that may be on a card is limited only to the size of the card field, column one to column 72. The simulator is so designed to ignore any quintuple or quintuple element that may appear beyond column 72 on the card. Anything that does appear beyond the limits will be shifted right one word on the output list. Therefore it is not advisable to put too many quintuples on a card. In most cases, four quintuples per card will give the User adequate spacing without overflowing the card.

The quintuple list must be followed by the word \textbf{DONE}. This will signal the completion of the quintuple list of User's Turing Machine. \textbf{DONE} may appear anywhere on the card between column one and column 72. It may be on the same card as the last of the quintuples or it may be on a separate card. However placed, it must
be preceded by a blank and it must be the last thing on the card.

The Exec card will follow the card with the word DONE. The only exception to this requirement is the Save file function which will be discussed in the section concerning the Turing Machine on file.

The Exec card signals the simulator that the builder has finished its operations and preparations for execution are made. The Exec card consists of the word EXEC followed by an optional parameter string. If the parameters are not needed, the word EXEC must be followed by the word DONE. The optional parameters are: Q, C, I, S, T and IP. They must follow the word EXEC and be separated by at least one blank.

The Q-parameter: The Q-parameter indicates the quintuple label used for the beginning of the execution of the User's Turing Machine. The format for the Q-parameter is Q=xx, where xx is any positive decimal number, or any quintuple label. If the User does not use the Q-parameter, the starting quintuple label will be one.

The C-parameter: The C-parameter specifies the column the User wishes to begin scanning. The column number will be the same column number on a card. The format for the C-parameter is C=xx, where xx is some decimal number either positive or negative. If the User does not specify the starting column, the starting column will be zero.

The I-parameter: The I-parameter specifies the print interval
for the output of the complete configurations. This allows the User
to output none, any or all of the complete configurations. The format
for the I-parameter is \texttt{I=xx}, where xx is some positive decimal num-
ber. If the I-parameter is not used, the print interval will be zero
and only the terminating complete configuration will be printed.

The \texttt{S}-parameter: The \texttt{S}-parameter is designed for the place-
ment of the Information Part, if there is an Information Part, in the
Information Tape. If the User wishes to specify the field for the be-
ginning of the Information Part, the \texttt{S}-parameter is a necessity. The
Information Part will be stored as it appears on the card starting at
the column specified by the \texttt{S}-parameter. The format for the \texttt{S}-
parameter is \texttt{S=xx}, where xx is some decimal number either positive
or negative.

The \texttt{T}-parameter: The \texttt{T}-parameter is used to signal the
simulator that this Turing Machine writes another Turing Machine
and wants the second Turing Machine executed. If there are no
errors in the first Turing Machine the second Turing Machine will
automatically be resubmitted to the simulator for building and exe-
cution. The format for the \texttt{T}-parameter is simply \texttt{T}.

The \texttt{IP}-parameter: The \texttt{IP}-parameter is used to signal the
simulator that an Information Part follows the Exec card. If the
User has a specific Information Part to be worked upon by the Turing
Machine, the \texttt{IP}-parameter must be used. The \texttt{IP}-parameter must
be the last parameter on the Exec card. If the Information Part is a blank tape, the IP-parameter must be omitted and the word DONE must be used. The format for the IP-parameter is simply IP. Either the IP-parameter or the word DONE will signify the end of the parameter list and will start the execution of the Turing Machine.

Examples of Exec cards:

```
EXEC C=25  S=189  I=5  Q=3  IP
EXEC   C=2   S=-65  Q=7   I=1   DONE
EXEC   DONE
EXEC   C=30  S=-30  T   DONE
EXEC I=6   C=5   IP
```

If there is an Information Part, it must follow the Exec card. It can start in any column or in the column specified by the S-parameter. It can take up the entire card and then continue on as many cards as are needed. The only limit to size is that the Information Part cannot exceed 4000 characters. Also if there is an Information Part, it must be followed by an end of file card. This will signify the end of the Information Part has been reached. If the Information Part is the blank tape, the end of file card is not used. The User may then follow the end of file card with another Turing Machine if desired, or a Logoff card to terminate the run. In the case of the machine working on a blank tape, the same order is followed with the exception of the end of file card, i.e. the next Turing Machine
may follow directly after the Exec card.

There are two forms of terminating messages used when the execution has been completed. When the User's Turing Machine has completed the work upon the Information Part as dictated by the Turing Machine itself, the message \textsc{Column} = xxxx will be printed, where xxxx is the column being scanned. This message will be followed by either the message \textsc{Missing Quintuple} = zzzz v, or the message \textsc{Quintuple} = zzzz v, where zzzz is the quintuple label and v is the symbol being scanned.

\textbf{Column=:} The column message indicates the column being scanned in the Information Part when the Turing Machine terminated its work upon it. The column is further indicated by an up-arrow, (↑), on the complete configuration. In the case of an error, the column indicator will allow the User to resubmit the final complete configuration as the Information Part, beginning at the column indicated. This will be feasible in the event the Information Part is lengthy and requires a great deal of work to be completed.

\textbf{Missing Quintuple=:} When the Information Part demands the use of a quintuple address that the simulator cannot locate, the above message will follow the column message. The first word following the message indicates the quintuple label, and the second word indicates the symbol being scanned. Again this indicates that the quintuple address is in fact missing from the quintuple list. Occasionally, but seldom, the printed list will have the quintuple
deemed missing, but in fact the quintuple is missing from the internal quintuple list and must be resubmitted into the list before operation on the Information Part will continue properly. This has occurred for the Author only in the case when several errors were already present in the Turing Machine.

**QUINTUPLE=:** When the work on the Information Part has been completed, in most cases successfully, the above message will be printed. Again the first word following the message will be the terminating quintuple label and the second word will be the symbol being scanned. If both words are zero, the machine has not encountered any execution errors, and the Turing Machine has worked correctly as designed by the User. The complete configuration following the message is the answer the Turing Machine has reached. If the answer is incorrect, the Turing Machine design is logically in error.

**Complete Configurations:** The intermediate and final complete configurations will be printed either starting at the column indicator, (↑), if it precedes the Information Part, or the Information Part, if it precedes the column indicator. The output line will consist of 24 words or 96 characters per line. The output line with the column indicator will also contain the quintuple label, the symbol being scanned and the column in which the indicator appears, respectively. The column indicator may not be pointing to the same character that appears as the symbol being scanned. This indicates that the quintuple
address at the end of this line has stored the replacement symbol in
the Information Part. The intermediate complete configurations are
used to follow the actions of the User's Turing Machine and are help-
ful in showing the strategy of the machine.

The Turing Machine on File

There are four actions the User has at his disposal making use
of files. The actions are desirable in that they simplify the work
with the Turing Machine. By saving any Turing Machine on the first
run, the User can get, correct and/or destroy any Turing Machine
quickly and easily. In most cases, once the Turing Machine is saved,
the User will find that the use of Teletype will be a great advantage as
a time saver for testing and simply using the machine.

Saving a Turing Machine: In order to save any Turing
Machine, insert a card before the Exec card with the following for-
mat: /SAVE Name. The slash must precede the word SAVE and the
name of the machine must be spaced by at least one blank. The name
need not be the same name given to the Turing Machine, but must
be of the same format. If the User does not use the same name, the
Turing Machine can only be referenced by the name under which it
was saved.
Example:  

```
7 8 *TURM
Name
  .
  .  DONE
/SAVE Name
  EXEC  C=20  Q=5  IP
  xxxxxxxxxx
```

The Turing Machine will be saved under the specified name whether or not any errors have occurred, assuming the errors are not terminating errors. Once the Turing Machine has been saved, the building process will no longer be used. Thus it is not necessary to resubmit the name or the quintuple list, and these may now be discarded from the User's deck. If the name used to save the User's Turing Machine has already been used, the message **FILE ALREADY EXISTS BY THIS NAME** will be printed and the Turing Machine will not be saved, but the simulator will proceed to execute the Turing Machine.

**Retrieving a Turing Machine:** To retrieve any saved Turing Machine, the file call takes the form: `/GET Name1`. The word `GET` must be preceded immediately by the slash, and separated from the name by at least one blank. The file call card must again precede the Exec card, but when retrieving a saved Turing Machine, the quintuple list is not needed.

Example:  
```
7 8 *TURM
/GET Name1
EXEC  C=3  Q=12  I=7  IP
  xxxxxxxxxx
```

```
77
88
```
If the name has not been saved, the simulator prints the message
**FILE NAME NOT IN DIRECTORY** then ignores everything up to the
next end of file card and searches for another Turing Machine to
process.

Correcting a Turing Machine: If the Turing Machine has been
saved, and yet there are some quintuples to be changed or corrected,
then the file function /CORRECT Name2 is required. Again the
function word CORRECT must be immediately preceded by the slash
and separated from the name by at least one blank. The correct card
must be followed by a card with the word REPLACE or the word ADD,
but not both. This word is in turn followed by the quintuple changes.
The quintuple address will indicate which quintuple needs the change,
and the replacement conditions will be substituted for the erroneous
conditions in the quintuple list. The list of quintuples used for cor-
rection must have the same format as the regular quintuple list,
and again followed by the word DONE. The Exec card will follow in
the same manner as above. The corrected Turing Machine will auto-
matically be saved under the same name. The only quintuple needed
in the list are the correction quintuples. In the case of a multiply
defined quintuple, there is no need for the correction unless the re-
placement conditions are erroneous.
Again if the name used for the correction is not a saved Turing Machine, the message **FILE NAME NOT IN DIRECTORY** is printed and the simulator will search for the next Turing Machine.

Destroying a Turing Machine: To destroy a saved Turing Machine simply use: /DESTROY Name4. In this case there is no need for an Exec card or an Information Part. Once the machine is destroyed, there is nothing more to be done with it. It can not be retrieved or used and if needed must be saved again. If the name of the Turing Machine has not been saved, the message **FILE NAME NOT IN DIRECTORY** will be printed and the simulator returns to process the next Turing Machine.

Example:  

```
7 *TURM
/DESTROY Name4
```

---

A Turing Machine Writing Another Turing Machine

When the User decides to have a Turing Machine write a second Turing Machine and execute it, he must use the T-parameter on the
Exec card. In order for the desired action to take place, the second Turing Machine when completed must be of the same format as the first. Thus the second Turing Machine must have a title and a quintuple list ending with the word DONE. The remaining functions follow normally. The title must be of the same format as that of the first, but need only be separated from the quintuple list by at least one blank. The quintuples must be of the same format, but the second Turing Machine must use a period, (.), in place of the dollar sign, ($). The Exec card and Information Part as well as the save function if used, need not be written by the first Turing Machine, but must follow the second Turing Machine as if the simulator were running for the first time.

Example:

```
7 *TURM
  Name5
  .
  .
  . DONE
EXEC Q=5 I=4 T DONE
/SAVE Name6
EXEC C=6 S=-459 I=10 IP
 xxxxxxxxxxxxxxxxxxxxx
77
88
```

For the first run of any Turing Machine, it is advisable to use Batch, unless the Turing Machine is very short. Teletype may cause a loss of time on the first run. In particular, when a line of data is
read, it is immediately printed. On Teletype, this redundancy wastes a great deal of time. Also when the User desires to output some of the intermediate complete configurations, the output line will use one and a half Teletype lines with the quintuple address and column being scanned in the second half of the second line. Finally the up-arrow, (†), used as a column indicator for the Information Part, will be separated from the output line it is scanning by at least one full line. At times the column indicator will appear to be scanning the quintuple address or the column number being scanned. Thus the User must remember that the up-arrow is pointing to a character in the second line above the actual up-arrow. The actual output of the intermediate complete configuration also seems to use more time on the Teletype.

Subsequent runs on the Teletype, especially after the Turing Machine has been saved, eliminates a great deal of work. If the intermediate complete configurations are few or are not printed, the Teletype becomes a very useful tool. The User will receive the results much faster. He will also be able to correct or destroy the Turing Machine quickly.

The Turing Machine used on Teletype requires the same format as the Turing Machine used on Batch. However, each line of input must be followed by using the return key and the line feed key. Instead of the end of file card, a Control W is used.
Example:  
	*TURM (ret)
Name7 (ret) (lf)
.
.
. DONE (ret) (lf)
EXEC C=45 Q=3 IP (ret) (lf)
xxxxxxxxxxxxxxxx (ret) (lf)
( Control W )

The file functions used on Teletype are also used in the same way as the functions used in Batch.

Example:  To retrieve a Turing Machine.

	*TURM (ret)
/GET Name8 (ret) (lf)
EXEC I=5 S=-70 DONE (ret) (lf)
/GET Name9 (ret) (lf)
EXEC C=10 Q=4 IP (ret) (lf)
xxxxxxxxxxxxxxxxxxxx (ret)
( Control W )

To correct a Turing Machine.

	*TURM (ret)
/CORRECT Name10 (ret) (lf)
REPLACE 1 D 6 X 0 DONE (ret) (lf)
EXEC C=4 I=6 DONE (ret) (lf)

To destroy a Turing Machine.

	*TURM (ret)
/DESTROY Name11 (ret) (lf)
( Control W )

In each case, the simulator will print the ending message and the final complete configuration. Then the simulator will search for the next Turing Machine to process. The only exception is the destroy function. When the work has been completed the User must follow the final routine with a Logoff.
In using any of the structures above, the User must type slowly after summoning *TURM. At times by typing too fast, the message **PARAMETER ERROR** will be printed. If this happens more than twice, it is advisable to simply Logoff and begin again. Also when the User intends to make corrections, he must take great care in using the required format for the corrections. If he does not, he will undoubtedly destroy part of the saved Turing Machine and will need to revise the entire Turing Machine on file.

**Errors**

**END OF DATA:** End of Data has been reached, terminating the simulation.

**FILE ALREADY EXISTS BY THIS NAME:** The name of this Turing Machine has been used for another Turing Machine and cannot be used to save this Turing Machine.

**FILE NAME NOT IN DIRECTORY:** The name of this Turing Machine cannot be found in the directory, this machine may not be saved.

**ILLEGAL CONTROL STATEMENT:** One of the parameters is not in the required format on the Exec card.

**ILLEGAL FILE CALL STATEMENT:** The file function is in some way erroneous and must be corrected.
**ILLEGAL QUINT LABEL:** The quintuple label is not within the range 1 to 1023, or the quintuple label is not a digit.

**ILLEGAL REPLACE, LABEL:** The quintuple replacement label is not a digit, or the digit is not in the range 1 to 1023.

**ILLEGAL TITLE OR FILE CALL:** The title card or file call is in error. A character in the title is not a letter or digit; the title card is missing; or the file function requested is in error.

**MUTLIPLE DEFINED QUINT:** This quintuple address has been encountered previously and is therefore ignored.

**NO DONE CARD:** The word **DONE** was not at the end of the quintuple list; or the word **EXEC** was erroneous. This will be followed by the message **ABNORMAL END** and the simulation is discontinued.

**NO EXEC CARD, NO EXECUTION:** The Exec card is missing.

**TOO MANY ERRORS:** More than 30 errors have been encountered and the simulation is discontinued.
M1
Is TM Flag Set?
Yes
Adjust Column Counter
No
GO TO FETCH

M2
Print a Blank Line
Print "ILLEGAL TITLE OR FILE CALL"
End of File
STOP2
CALL READ
Normal
GO TO START

M3
CALL PSYM
SYM → Title Area
CALL FILES
CALL INCHAR
CALL CLAS
digit
Print a Blank Line
Print "NO EXECUTION"

CALL PSYM
Is word "EXEC"?
No
Yes
GO TO EXEC
Search for beginning of the Information Part

Was the file Already Equiped ?

Yes

Unequip the File

No

Search for beginning of the Information Part

Found

Store Blanks in TDT

Store two words of Information Part in TDT

Has end of Info Part been reached ?

No

18 words been processed ?

No

CALL STMF1

STMD

Yes

CALL STMF1

STMD

CP

COL + 1 → COL
STMD

Write a File Mark on the File

Rewind the File

Change Input unit

Set TM Flag

Stores Zeroes in the old Builder Area

GO TO START

STMF1

Load a Character from TDT

Replace Period with Dollar sign

Yes

Is Character a period?

No

Is Character at end of TDT?

No

Increment TDT column index

RETURN

I PLC

Reset Input unit

RETURN
ERROR

Store Line number in error table

Store error code in error table

Store quintuple in error table

Increment error pointer

More than 30 errors?

Yes

CALL ERRPRINT

No

Decrement EM

CALL INCHAR

Increment Column Counter

Load next character

Is Character a Blank?

Yes

No

EC

Is EM = 0?

Yes

RETURN

No
ERRPRINT

Are there any errors?

No → RETURN

Yes → CALL PAGE

Store Line Number in error Line

Store error message in error Line

Store Quintuple in error Line

Print error Line

Was this the last error?

No → Increment the error table pointer

Yes → Print a Blank Line

RETURN
FETCH

Was LCR a Blank?

Yes

CALL INCHAR

CALL DECONV

CALL INCHAR

"And the character with Quint.

No

F3
digit

CALL INCHAR

CALL CLAS

letter

Set error code EM = 4

CALL PSYM

CALL ERROR

F3

A2

RETURN TO EDITS

Set Done Flag

Is Edit Return Flag Set?

Yes

No

Is word "DONE"?

No

Yes

CALL IPLC

CALL ERRPRINT

CALL INCHAR
CALL CLAS
CALL PSYM

GO TO EXEC

Is word "EXEC"?

CALL SERCH

Set error code EM=3
CALL BICONV
CALL ERROR

CALL ERROR

F3
FETCH2
CALL INCHAR
CALL PSYM
CALL CLAS
CALL DECONV
CALL INCHAR
CALL INCHAR
CALL BICONV
Set error code EM = 2
CALL ERROR
Store word "STOP" in RSTAT
CALL STORE
F 3
RETURN TO EDITS
STORE

- Store Quintuple in Quintuple List
- Increment Quintuple List Pointer
- Increment TOTAL

RETURN

PAGE

- Eject a Page
- Print Heading
- Reset Page Line Counter to zero

RETURN

PRNTB

- Print CARD

- Is Page Line counter > 56

CALL PAGE

- Increment Page Line Counter

RETURN
READ

Is Status at End of Data?

Yes
Print "END OF DATA"

No
Blanks → Card Area

Terminate

Read a card → CARD

File Mark?

Yes
RETURN END OF FILE

No
Increment Line Counter

CALL BICONV

Separate Column 73-80 from rest of Card

CALL PRINTB

RETURN NORMAL
INCHAR

Increment Column Counter

Is Column $\geq 72$?

No

CALL READ

Reset Column Counter to zero

Is character at COL a Blank?

No

Is character a Dollar Sign?

No

Character $\rightarrow$ LCR

RETURN

Yes

End of File

CALL ERRPRINT

Normal Return

Print "NO DONE CARD"

Print a Blank Line

Print "ABNORMAL END"

GO TO START
PSYM

Blanks
→ SYM

Load Character

CALL CLAS

Special Character

RETURN

Letter or digit

Store character
Left-justified in SYM

Increment Column Counter

CLAS

Load Character

Search table for appropriate index

Add index to RETURN Address

RETURN
AG

Is Print Interval = 0 ?

No

Is Print Counter = 0 ?

No

Decrement Print Counter

Load MC

Is MC \geq 2 ?

No

Add MC to Column Counter

Is Stop Flag Set ?

No

A4

Yes

CALL INFOUT

AS

Is error Flag set ?

No

Store symbol being scanned in output message

CALL BICONV

Print "MISSING QUINTUPLE = "

CALL INFOUT

GO TO START

Yes

Store -1 in MC

Yes

CALL INFOUT

Go To STM

No

GO TO START

AN

Print "ILLEGAL CONTROL STATEMENT"

GO TO STOP2
INFOIN

Check End of Data?

Yes

Print "END OF DATA"

No

Store Blanks in CARD

Terminate

Read data into CARD

End of File?

Yes

Return End of File

No

Print the Card

Return Normal
SEARCH

Set Pointers

Is D-Pointer at Quintuple?

Yes

Is D-pointer = U-pointer?

No

Decrement D-pointer

Is U-pointer at Quintuple?

Yes

Return Found

No

Is U-pointer = D-pointer?

Yes

Store Bottom Pointer in U-Pointer

No

Increment U-pointer

Down

Is U-pointer = Top-pointer?

Yes

Store Bottom Pointer in U-Pointer

No

Increment U-pointer

Down

Return Not Found
Set Flags and Counters

At Beginning of Info Part?

Has Column been reached?

Store 2 words of Info Part in output line

Have 3 Blank words been stored?

Is Column flag set?

Increment Output line word counter and Info count

Print output line

Reset Print Interval Counter

RETURN

Set flag to set column flag, FIT

Save Column counter for Info Part

Is FIT set?

CALL WOUT
CALL BICONV

Store column counter in Output Line

CALL BICONV

Store symbol being scanned in Output Line

CALL WOUT

Store Quint. Label in Output Line

CALL WOUT

Store (↑) in column being scanned

CALL WOUT

I4

WOUT

Print Output Line

Store Blanks in Output Line

Reset column counter for Info Part

Set column flag

RETURN

I2
FILES

Load File Function in FSYM

CALL RWSRCH

Found

Load index from search

Go to Appropriate Routine

CR TCF Correct

Save SVEF

Destroy DESF

Get GF

UEFILE

Unequip Logical Unit 1

RETURN

UEFL3

Unequip Logical Unit 3

RETURN
CALL DIRECT

Found

Load Corresponding Systems Name

Equip Name to Lun 1

Is Lun 1 already equipped?

Yes

CALL UEFLE

Read Number of words on file

Read file into memory

CALL EDITS

Delete corresponding systems name from systems directory

Equip Lun 2 to a file

CR1

Not found

NF

CR1

Is Lun 2 already equipped?

Yes

CALL UEFLE

Print number of words on file

Print Turing Machine on file

Write File Mark on the file

Rewind Lun 2

Save Lun 2 under Systems corresponding name

Unequip Lun 2

CALL UEFL3

RETURN

Unequip Lun 2

CR2
CALL DIRECT

Found

Load corresponding systems name

Equip name to Lun 1

Is Lun 1 already equipped?

Yes

CALL UEFL

No

Read number of words on file

Read file into memory

CALL UEFL

CALL UEFL3

RETURN
CALL DIRECT

Search Directory for a Blank word

Store Turing Machine name at end of Directory

Load corresponding systems name

Increment Directory name counter

Equip Lun 1 as a file

Is Lun 1 already equipped?

Load corresponding systems name

Print a Blank Line

Print "FILE NAME ALREADY EXISTS"

CALL UEFL3

RETURN

CALL UEFL3

Rewind Lun 1

Save File under systems name

CALL UEFL

Rewind Lun 3

Write number of names on file

Write Directory on file

Write File Mark on file

Rewind Lun 3

CALL UEFL3

RETURN
EDITS

Set Edit
Return
Flag

CALL INCHAR

Store Blanks
in FSYM

CALL PSYM

CALL RWSRCH

CALL FETCH3

CALL SERCH

CALL FETCH2

CALL UEFLE

CALL UEFL3

GO TO STOP2

Store RSTAT
correction in
Quintuple List

Store Quintuple
at end of
Quintuple list

REP

RETURN

Is Done
flag set?

Yes

Not found

Found

Not found

Found

REP
DIRECT
Equip Lun 3 to DIRTJ

Read number of names in Directory

Is number = 0

Yes
Load word in FSYM

RETURN
Not found

No
Read Directory into memory

Search Directory for Turing Machine name

Load index for corresponding systems name

RETURN
Found

RETURN
Not found

RWSRCH
Load word in FSYM

Not found

Search RWTAB for word

Found
Add one to Return Address

RETURN

RETURN

RETURN

RETURN

RETURN

Print a Blank Line

Print "ILLEGAL FILE CALL"

Not found

Is word "ADD" or "REPLACE"?

Yes
Set jump index

RETURN

No

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN

RETURN
APPENDIX IV

PROGRAM DEFINITIONS

ABEND: An error message.

BICONV: Routine to convert binary numbers to decimal.

BLANK: Two blank words.

CARD: Location for one line of builder input.

CLAS: Routine to determine if a character is a special character, digit or letter.

COL: In the builder routine, the column being scanned on CARD. In the execution routine, the column being scanned on the Information Part.

COLM: The temporary location for the starting column used in the execution routine.

CRTCF: Routine used by FILES to correct a Turing Machine.

D: Done flag used to indicate the end of the building process for the EDITS routine.

DAT: Location in HEADING for the date the Turing Machine is being processed. Date is stored in the form: MM/ DD /YY.

DECONV: Routine to convert decimal numbers to binary.

DESF: Routine used by FILES to destroy a Turing Machine.

DIGIT: The temporary location for the number being converted.
EDITS: Routine used by CR TCF to correct quintuples.
EDRET: Flag to indicate a return to the EDITS routine.
EM: Location for the error code in the builder routine.
ERRLINE: Location for the line number when outputting builder errors.
ERRMESS: A table for error messages.
ERROR: Routine to process builder errors.
ERRPRINT: Routine to output builder errors.
ERPTR: Pointer for the storage of errors in the error table.
ERRTAB: Storage area for the builder errors.
EXEC: Routine to execute a Turing Machine.
FETCH: Routine to build and store a Turing Machine.
FETCH2: Routine to analyze and store the replacement conditions.
FETCH3: Routine to process a file function.
FIT: Flag to indicate that the column being scanned has been reached in the present output line.
FNAME: Location for the index to the list of corresponding systems names for saved Turing Machines.
FND: An error message.
FNPR: Number of words in the *TURM directory of Turing Machine names. Two words of storage are used for each Turing Machine name.
FPTR: Pointer to the reserve word table indicating the file function desired.

FSYM: Location for the file function desired.

FTMP: Location for the *TURM name equivalent to the Turing Machine name used by the User.

GF: Routine used by FILES to retrieve a Turing Machine from storage.

HEADING: Storage location for the word *TURM, the title, date and time of the run.

HIGHMEM: Systems indication of the top of usable memory.

IFCS: An error message.

INCHAR: Routine to locate the next non-blank character on a card or line.

INFOIN: Routine to input the Information Part.

INFOOUT: Routine to output the complete configurations.

INPART: Location for the last half of the Information Tape.

INPART1: Location for the first half of the Information Part.

INSTOR: Location for the index used in storing the Information Part into the Information Tape.

INTER: Location for the print interval for the complete configurations.

INUNIT: Location for the logical unit number used for input.

INUNT1: Temporary location for the regular input unit.
IOLUNS: A *SYSLIB routine to determine the I/O units.

IPLC: Routine to change the input unit back to the original input unit.

IOT: The temporary location for the Information Tape column counter.

IOT1: The complement of the column being scanned of the Information Part. The complement does not exceed 96, the length of one output line.

ITAB: Table of names used for the systems directory to save the Turing Machine.

JTAB: A jump table for the jump to the routine handling the desired file function.

LCR: The last character read by INCHAR.

LINE: Two words used in output for the line number.

MAIN: Routine to initialize *TURM.

MC: Temporary location for the move code.

MESS: Location for output of the builder errors.

NOC: An error message.

NODONE: An error message.

NUMB: Temporary storage location for the conversion of a number.

OCOL: Storage location for the column number indicating the symbol being scanned.
OQNT: Storage location for the last quintuple label used by the complete configuration.

OSYM: Storage location for the last symbol being scanned for the complete configuration.

OUT: Storage area for the output of the complete configuration.

OUT1: Two words of blanks used to eliminate the problem of carriage control for the output of the complete configurations.

OUTUNIT: Location for the logical unit number used for output.

PAGE: Routine to eject a page and print the Heading.

PCIS: An error message.

PCOL: Column of the symbol being scanned used in output when terminating.

PEOD: An error message.

PFEX: An error message.

PIC: The print interval counter.

PLC: Page line counter limited to 56 lines of builder output per page.

PMQT: Location for a missing quintuple address.

POUT: Temporary storage location for the Information Tape index.

PRINTB: Routine to output the quintuple list.
PQNT: The location for the terminating quintuple address.

PSYM: Routine to pack letter-digit strings left-justified.

PTHR: An error message.

PTME: An error message.

QNPTR: Location for the top of the quintuple list.

QUINT: Location for the quintuple address.

READ: Routine to read a card or line of data into memory.

REWIND: Systems function to rewind a file.

RQT: Location of the converted number upon return from the conversion routine.

RSTAT: Location for the replacement conditions.

RWTAB: A table of reserve words containing the file functions.

SERCH: Routine to search the quintuple list for a specific quintuple.

SFLAG: A stop flag.

SIGN: Temporary location for the sign of a number used during conversion.

STAD: Address of the bottom word of the quintuple list.

STATE: Two words of storage used for the quintuple when storing it in quintuple list.

STATUS: Systems function to determine the status of the input unit.
STEP: Pointer used to indicate the end of the search of the quintuple list.

STM: Routine to set up for input a Turing Machine written by another Turing Machine.

STMF1: Routine to write a second Turing Machine on a file.

STOP2: Routine to clean-up memory by MAIN.

STORE: Routine to store a quintuple in the quintuple list.

STPTR: Pointer for the quintuple in the quintuple list.

STRT: Address of the bottom word of the quintuple list.

SVEF: Routine used by FILES to save a Turing Machine.

SYM: Location for a packed letter-digit string.

T1: Location for the line number.

T2: Flag for a Turing Machine written by another Turing Machine.

TCOL: Flag indicating the output line containing the column being scanned has been printed.

TDATE: Routine to determine the date of the run.

TDT: Temporary storage area for the Turing Machine written by another Turing Machine.

TEFL: Flag for an erroneous stop condition.

TIM: Location in HEADING for the time the Turing Machine is processed. The time is stored in the form: HHMM.
TIMER: Routine to determine the time of the run.

TM: Flag to indicate the need for execution of a Turing Machine written by another Turing Machine.

TOTAL: Location for the number of words in the quintuple list, two words per quintuple.

TITLE: Location in HEADING for the title or name of the Turing Machine.

WFM: Systems function to write a file mark on a file.

WOUT: Routine to print a complete configuration.
RETURN 2. LETTER
CHECK FOR A -/
RETURN 0.
SPECIAL CHARACTER
RETURN 1.
DIGIT
CHECK FOR -EXEC-
PRINT A BLANK LINE
STORE LINE NUMBER IN ERROR TABLE
PRINT TOO MANY ERRORS-
CHECK TO SEE IF THERE IS A LIST OF ERRORS TO BE PRINTED OUT
STORE ERROR MESSAGE, LINE NUMBER AND ERRCODE IN A WORD CALLED MESS
PRINT A BLANK LINE
PRINT -INVALID TITLE OF FILE CALL
PRINT -NO EXECUTION-
PRINT A BLANK LINE
A SECOND TIME
UNIQUE IT

LOAD IMPORT1.2
LOCATE THE BEGINNING OF
THE INFORMATION PART

CHECK FOR END OF INFO PART

BLANKS IN TOT AREA

PICK UP FIRST 18 WORDS
OF INFO PART

CHECK FOR END OF INFO PART

CHECK FOR ENO OF TOT

REWIND THE FILE

CHANGE INPUT UNIT TO

WRITE 20 WORDS ON FILE FOR INPUT

PF STORE INPUT UNIT

RETURN
STAG
LINE
LOAD CARD+18
STA CARD+19
RTJ PRINTB
ENA 1
ANA READ
SNA READ
END READ
ENM PFMD
ENO \\ WRITE, I
OUTUNIT
SNUM
PAGE UJPN
ENA HEADING
END 26
WRITE, I OUTUNIT
ENA 0
STA PLC
UJP PAGE
PRINTB UJPN
ENA LINE
END 21
WRITE, I OUTUNIT
LDA PLC
ASG 56
UJP **1
STA PAGE
INA 1
STA PLC
UJP PRINTB
PSYM UJPN
CTI PD2, 2
LDAQ BLANK
STAD SYM
ENI *7, 2
LACH CARD+1
RTJ CLAS
UJP ES
NOR INT 1, 1
SACH SYM+7, 2
AIJ MSL
UJP SYM
ES LCHO SYM
ENI *7, 2
UJP PSYM
CLAS UJPN
STI CLR1, 1
T11 1
LACH CLR1, 1
SWX CLFX
T1A 1
T1A **1
CLXE UJPN
FLEN OCT 0110101, 0111011
OCT 01115300, 0
OCT 0223227, 0223227
OCT 0223220, 0
OCT 0223220, 0
OCT 0223203, 0

BICONV UJP **
SCHA SS
INA 1
SCHA SS
STA NUMB
OCT 0602522, 02220732
OCT 02220200, 0
ENO 4
PRINT END OF DATA
OSG S
STA **
ENK 0
ENA 0
UJP BOLD
BOL JWA TFN
SO SOCH **1
SHAD 24
BOL UJP BOLD, 1
ENA ANR
ENA 2BN
SS EXCH **
UJP BICONV
DECONV UJP **
SCHA FCH1
ENI 0.2
ENK 0
STA NUMB
STA SIGN
IGN RTJ FCH
ASG 10
UJP DG1
ASG 4BN
IGN UJP DGN
ENA = 0
STA SIGN
IGN UJP IN
STA DIGIT
LDA NUMB
MUA TCH
ANA DIGIT
STA NUMB
INT 1.7
RTJ FCH
ASG 10
UJP DG1
LDA NUMB
SCH SIGN
INI 1.1
UJP DECONV
FCH UJP **
LACH **1
INI 1.1
UJP FCH
NUM ASS 1
DIGIT ASS 1
TEN DEC 10
SIGN ASS 1
ZERO OCT 0
ROT ASS 2
MODERN MOD 4, NO DONE CARD
AHEAD NCD **, ABNORMAL END
END