United States Atomic Energy Commission

Research and Development Report

POGO
PSAG Manual

by

D. R. Fitzwater, E. H. Hietbrink,
M. M. Ledet, D. E. McFarland
and C. E. Runge

May, 1964

Ames Laboratory
at
Iowa State University of Science and Technology
F. H. Spedding, Director
Contract W-7405 eng-82
This report is distributed according to the category Mathematics and Computers (UC-32) as listed in TID-4500, April 1, 1964.

Legal Notice

This report was prepared as an account of Government sponsored work. Neither the United States, nor the Commission, nor any person acting on behalf of the Commission:

A. Makes any warranty or representation, expressed or implied, with respect to the accuracy, completeness, or usefulness of the information contained in this report, or that the use of any information, apparatus, method, or process disclosed in this report may not infringe privately owned rights; or

B. Assumes any liabilities with respect to the use of, or for damages resulting from the use of any information, apparatus, method, or process disclosed in this report.

As used in the above, "person acting on behalf of the Commission" includes any employee or contractor of the Commission, or employee of such contractor, to the extent that such employee or contractor of the Commission or employee of such contractor prepares, disseminates, or provides access to, any information pursuant to his employment or contract with the Commission, or his employment with such contractor.

Printed in USA. Price $2.00. Available from the
Office of Technical Services
U. S. Department of Commerce
Washington 25, D. C.
# CONTENTS

| ABSTRACT                          | 5 |
| INTRODUCTION                     | 5 |
| DESIGN PRINCIPLES                | 6 |
| GENERAL FLOW OF POGO OPERATION (Chart 01) | 8 |
| SNEAK (Chart 2)                  | 10 |
| POGO MODIFICATIONS AND ADDITIONS TO SYCA 3 (Chart 3) | 12 |
| POGO MODIFICATIONS TO SYCL2      | 30 |
| POGO MODIFICATIONS TO UPDAT      | 30 |
| OTHER MODIFICATIONS              | 31 |
| POGO FUNCTIONS                   | 34 |
| ADDITION OF FUNCTIONS            | 52 |
| POGINOUT 1401 PROGRAM            | 55 |
| ACKNOWLEDGMENTS                  | 56 |
| FLOWCHARTS                       | 57 |
IS-893
IS-893

POGO PSAG MANUAL

D. R. Fitzwater, E. H. Hietbrink, M. M. Ledet,
D. E. McFarland and C. E. Runge

ABSTRACT

The problem considered in this paper is the design and structure of the POGO monitor. The programs comprising the monitor and the necessary modifications to the compiler systems tape are discussed. Flow charts accompany the writeups.

INTRODUCTION

The problem considered in this paper is the design and structure of POGO (Programmer Oriented Generalized Operator). The format of this paper parallels that of a "programming systems analysis guide" since the paper is intended to supplement the "POGO" and "Compiler Systems" manuals in providing information on the internal logic of the POGO system. This information is to be used by personnel in charge of maintaining, designing, or updating the system. It is hoped that the design of the system will encourage its expansion.

For all but very minor changes to the system it is essential to possess the compilation listings of POGO and the Compiler Systems and
a general knowledge of the 7070 Compiler Systems control routines. Information on the systems control routine is available in the following manuals:


None of the information in this paper should be required knowledge for one to program or operate within the system.

Throughout this paper, except in flow charts, the following convention has been followed: names and labels belonging to POGO are capitalized, while names and labels belonging to the IBM 7070 Series Compiler System are capitalized and underlined.

DESIGN PRINCIPLES

A tape oriented computer as fast as the IBM 7070/7074 must be considered as a tape to tape processor. Communication with the computer may be made through tape, console card reader, and console typewriter. Only tape communication offers speeds that are compatible with the internal processing speeds.

Although some installations will have a relatively small number of long running jobs, such is not the case here. Manual setup time for
modest length jobs can easily produce substantial delays in processing. This is particularly true if tapes must be mounted or demounted.

Manual job setups, while the machine is waiting, are not only slow but are peculiarly prone to error because of the press of time. The same is true for programmer decisions during the running of a program.

Any substantial amount of communication must be through tape. Tapes are prepared and dumped through a peripheral IBM 1401. The tape manipulation task becomes prohibitive if several tapes must be manually manipulated for each job unless the jobs are relatively long running. The natural consequence is that all but the long running jobs must be batched on a few tapes. The longer jobs may also be batched but the necessity is not as strong.

In order to eliminate the difficulties mentioned above and to decrease the otherwise large operating staff it is standard practice to have the computer assist the operator by means of an operating program. All of the routine, and some of the not so routine, operating decisions can be made by the computer with little or no intervention. The operator can spend his time monitoring for unusual situations, and routine operations are performed more reliably.

The POGO system was created with the following goals in mind:

1. Job batching to minimize tape handling and job setup time.
2. Accounting for time used with an internal clock.
3. Configuration flexibility to adapt to hardware changes.
4. Running user's program with minimum restrictions.

5. Facilities for effective operator intervention in unusual cases or where desired.

6. Standard diagnostics if trouble is encountered.

7. Preset diagnostic options for expected or unexpected troubles.

8. Facilities for common manipulations.


10. Standardization of messages and halts for operator control.

GENERAL FLOW OF POGO OPERATION (Chart 01)

The POGO system exists as a graft on to the IBM Compiler Systems Tape and operates in conjunction with it.

Block 01: The call for POGO may be made by loading (with the load button) a POGO restart card from the console card reader. The restart card will have the restart parameters punched into word 7. Thus, by having a few standard cards pre-punched, the various restart options may be easily obtained.

Block 02: The call for POGO may be made by the operator placing the address of the restart routine in the instruction counter and pressing start (possibly in the diagnostic sense mode). The restart routine will simulate the restart card of Block 01. Block 01 procedure must be used if the restart routine is damaged. The advantage of the Block 02 routine is that diagnostics unobtainable after computer reset may be obtained by SNEAK.
Block 03: The restart routine will be planted upon entry of the system. The loaded program may return control to POGO by transferring control to the restart routine, after planting the desired restart parameters in the COMREC. Note that the restart routine and the COMREC are the residual of the POGO system which should remain untouched in the core with the users program. The locations (25) should not be used for other purposes.

Block 04: SNEAK is loaded from the front of the systems tape. SNEAK is a multiphase program which will process the restart request, provide requested diagnostics, and arrange to save the core about to be overwritten by .SYCL1 and .SYCA3. The exact tasks performed depend upon the restart options in the restart request. A normal restart request will not produce any diagnostics or dumps. SNEAK will place its output (if any) onto the specified list and deck tapes. These are specified in the restart parameters and are normally the system list and deck tapes. SNEAK then calls BOOTSTRAP.

Block 05: BOOTSTRAP is the original Compiler Systems loading routine which has not been modified. This routine loads .SYCL1.

Block 06: .SYCL1 contains IOCS and subroutines for the use of locating and loading programs on the Compiler Systems Tapes. .SYCL1 also loads the systems COMREC and .SYCA3. The COMREC coding block has not been modified.
Block 07: The basic POGO system is contained as a graft onto .SYCA3. POGO will scan the initial input file for system calling cards and execute the desired POGO function. POGO input is always from the initial input unit. The desired POGO function may be in core with .SYCA3, or another coding block on the systems tape. POGO is designed in such a way as to make possible the easy addition of new POGO functions. The system is designed to grow and to adapt.

Block 08: POGO functions can be of a large variety e.g. a routine to type remarks or a routine to compile FORTRAN. These functions will eventually return control to POGO, perhaps through a users program by a restart or by a direct branch.

SNEAK (Chart 2)

SNEAK replaced BOOTSTRAP on the front of the Compiler Systems Tape. It will give one of four different types of restart. Type 1 is a normal restart code (90-99), no SNEAK action is taken. Type 2 is an abnormal restart code (61-79) with NO preparation for a core restore. This includes invalid alpha search, latch listing, and index words. Type 3 is an abnormal restart code 80-89 with preparation for a core restore. This type is the same as type 2 with the addition of a core restore. Type 4 is a neutral restart code 15.

During an abnormal restart SNEAK checks the core for alpha words containing invalid alpha combinations. These locations are made plus, and a list of them is dumped onto the LISTP along with the status of indicators, alteration switches, tape priority interrupt latches,
electronic switches, error latches, accumulators, index words, and a constant dump of locations 100-199 and 300-329.

Block 01: Save the accumulators, they may be needed later for output.

Block 02: Type restart parameters.

Block 03: If the DECKTP does not equal LISTP go to Block 05.

Block 04: Halt to allow operator to manually correct the restart parameters since the output and core restore can't be on the same tape.

Block 05: If normal or neutral restart, go to Block 07.

Block 06: Load PRINT, go to Block 08, where execution of PRINT begins.

Block 07. Load BOOTSTRAP, transfer to it.

Block 08: Write locations 15-199 and 300-329 onto the DECKTP.

Block 09: TM LISTP, write restart parameters.

Block 10: If restart class is in the 80's, go to Block 12.

Block 11: Skip over SNEAK ALPHA, go to Block 16.

Block 12: Load SNEAK ALPHA, go to it.

Block 13: Search for invalid alpha combinations in alpha words.

Block 14: Put out a table of the locations of the invalid alphas.

Block 15: Write restore on deck tape.

Block 16: Load MAIN, transfer to it. This is the section that dumps the diagnostics.

Block 17: Write status of indicators, latches, electronic switches, index words, and a constant dump locations 100-199, 300-329.
POGO MODIFICATIONS AND ADDITIONS TO .SYCA3 (Chart 3)

.SYCA3, including the POGO modifications and some POGO
FUNCTIONS, is loaded from the systems tape by .SYCL1. The POGO
COMREC is placed at the top of the expanded memory. In such a case,
POGO will run in expanded mode but will transfer to unexpanded mode
when control is given to .SYCA3.

Block 01: Set the output deck and list tapes for multifele runs into the
running time unit designations, PH3DECK and PH3LIST.
All compiler systems output will be produced on the multi­
file output units. Modifications to prevent rewinding of these
units must be made to .SYCL2 and .SYCL6.

Block 02: Modify initial input DTF to ignore segment marks, long and
short length records, and to branch to POGO end-of-reel
routine (Chart 6) when a tape mark is encountered.

Block 03: Recompute POGO pseudo check sum from portions of the
POGO COMREC to see if it has been damaged.

Block 04: Test check sum in POGO COMREC, if satisfactory, go to
Block 06, otherwise go to Block 05.

Block 05: POGO COMREC is damaged so type warning message to
operator and continue to Block 06.

Block 06: The restart class is examined for a 15 (double digit for a
period). If 15, then the restart is considered neutral and
LASTCLASS is not changed. Otherwise LASTCLASS is to be
changed.
Block 07: Set LASTCLASS to current restart class.

Block 08: Set restart class to Z(89). The restart routine will use this class when next entered unless a function or user program resets it.

Block 09: Transplant new versions of the restart routines to their operating positions.

Block 10: If POGO input cards are to be read from the console card reader, go to the card reader patch (Chart 11), otherwise go to Block 11.

Block 11: Use .SYCA3 facilities to obtain the next record from the initial input unit. A tape mark will cause branching to the POGO end-of-reel routine (see Chart 6).

Block 12: If entered, a tape mark was not found, so turn off TMSWITCH in end-of-reel routine. Two successive tape marks must be encountered to cause the end-of-reel routine procedure to close out the input file.

Block 13: If this record is identified with POGO in cols. 76-79, go to Block 16, otherwise go to Block 14. The class in col. 80 is ignored in this test.

Block 14: If this is the first non-POGO card encountered, go to Block 15, otherwise go get the next input record.

Block 15: Type SKIPMSG which identifies the first record skipped. This is of interest to the user whose processing of the input file was incomplete. This could occur because of machine, programmer, or operator errors.
Block 16: One or two change numbers may have been planted in the POGO COMREC. If so, go to Block 17, otherwise go to Block 19.

Block 17: Check waiting change numbers against sequence number in col. 71-75 of the calling card. This number was inserted by the IBM 1401 program POGOINOUT and serves to key changes to the input file. If this record is to be changed, go to Block 18.

Block 18: Type a message indicating that a change is expected and go to the standard interrupt procedure (see Chart 5) for the change. If change is ignored, go to Block 19.

Block 19: Compare calling card class to LASTCLASS. If calling class agrees or is blank, accept this card for further processing, otherwise get new record.

Block 20: Is calling class blank? If it is, go to Block 21.

Block 21: Reset LASTCLASS to blank.

Block 22: Is current record an accounting card?

Block 23: If yes, do accounting procedure (see Chart 7, POGO ACCOUNTING PROCEDURE), and go to Block 10 for new record.

Block 24: Is accounting switch on? This implies that current accounting information is in the POGO COMREC and that some account is held responsible for the time. If accounting switch is off, go to next input record, otherwise, accept calling card for execution.
Block 25: Type IDENTMSG identifying accepted calling card. This message contains the sequence number and the class.

Block 26: Look up OPCODE in POGO OPCODE TABLE. This table may be expanded to include new POGO functions.

Block 27: Is opcode in POGO OPCODE TABLE? If not, turn control over to RUNCHECK (Chart 8).

Block 28: Let SYCA3 look in its own opcode table (STCJ3).

Block 29: Is OPCODE in STCJ3?

Block 30: No. Type error message and go to standard interrupt procedure for operator action.

Block 31: Yes. Continue normal SYCA3 operations. If SYCA3 gets new initial input record, POGO will again take control at Block 11.

Block 32: Execute specified POGO function. This involves branching either to a specified function or to a routine which will load the specified function.

There are three RESTART ROUTINES (Chart 4) in POGO. One is used only if PROCESSADD EQUIP NO Option is used. For such an unexpanded (and presumably unexpandable) machine a restart routine and the POGO comrec are placed at:

9950-9964  Save area for index words 0-14
9965-9983  Restart routine
9968       Entry point to restart routine
9984-9989  POGO COMREC
The other two restart routines (used if PROCESSADD EQUIP YES) operate as a pair and are placed at:

9975-9983 Restart from unexpanded mode
19960-19974 Save area for index 0-14
19975-19983 Restart routine part 1
(1)9975 Entry point to restart routine
19984-19989 POGO COMREC
19990-19999 Restart routine part 2

The type of restart routine planted by POGO depends upon the PROCESSADD option. This restart routine (and the POGO COMREC) is the residual of the system which remains in the core during execution of a users program.

The purpose of the restart routines is to provide automatic facilities for the user to return control to POGO, or for the operator to do so through the console. If the machine is in expanded mode, the operator may call POGO by branching to 19975. If the machine is expandable, but not in expanded mode, the branch is made to 9975. If the restart routines have been damaged, the operator may call POGO by use of the console load cards (using the console load button). This procedure is slower and does not provide a complete set of diagnostics from SNEAK. All of the error indicators are lost and the first 15 index words will be invalid.

Block 01: This is the restart if expandable machine is in unexpanded mode. This simply secures entrance to the expanded mode restart routine.
Block 02: This is the restart if in expanded mode.

Block 03: Block further priority branches.

Block 04: Turn on additional storage switch.

Block 05: Make sign alpha of the restart parameters in the POGO COMREC.

Block 06: Set mask to inhibit interrupts, make restart parameters plus (SNEAKS way of knowing which restart routine was used).

Block 07: Wait for channels to become free.

Block 08: Initialize tape unit 10 for reading.

Block 09: Save index words 0-14 in upper memory to make room for SNEAK.

Block 10: Load SNEAK phase 1 from front of systems tape.

Block 11: Pick up restart parameters from POGO COMREC and plant in index word 6.

Block 12: Wait for completion of tape operation.

Block 13: Execute SNEAK phase 1.

The POGO COMREC is used to convey information between loadings of POGO. The COMREC has the form:

<table>
<thead>
<tr>
<th>CNTRL</th>
<th>9984</th>
<th>(or 19984)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>ACCNTPARAM</td>
<td>00, 00</td>
<td>For statistics</td>
</tr>
<tr>
<td>SNAPCODE</td>
<td>01, 01</td>
<td>Code for the SNAPSHOT routine</td>
</tr>
<tr>
<td>CHANGENO</td>
<td>02, 09</td>
<td>For console interrupt</td>
</tr>
<tr>
<td>ACCNTNAME</td>
<td>10, 19</td>
<td>Alpha account name</td>
</tr>
<tr>
<td>Field</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>-------------------------------------------</td>
</tr>
<tr>
<td>ACCNTNAME</td>
<td>20, 29</td>
<td>Numeric account name</td>
</tr>
<tr>
<td>ACCNTNO</td>
<td>30, 39</td>
<td>Account number</td>
</tr>
<tr>
<td>LASTCLASS</td>
<td>40, 41</td>
<td>Previous acceptable class</td>
</tr>
<tr>
<td>SEQNO</td>
<td>43, 44</td>
<td>Account sequence number</td>
</tr>
<tr>
<td>SCANLSTOPT</td>
<td>45</td>
<td>SCANDAL listing option</td>
</tr>
<tr>
<td>SCANFRCOPT</td>
<td>46</td>
<td>SCANDAL force option</td>
</tr>
<tr>
<td>STARTIME</td>
<td>47, 49</td>
<td>Filled by account procedure</td>
</tr>
<tr>
<td>ORIGIN</td>
<td>CNTRL 9984 (or 19984)</td>
<td></td>
</tr>
<tr>
<td>DA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>DECKCH</td>
<td>50</td>
<td>Current</td>
</tr>
<tr>
<td>DECKUN</td>
<td>51</td>
<td>Deck tape</td>
</tr>
<tr>
<td>DECKTP</td>
<td>50, 51</td>
<td>For POGO restarts</td>
</tr>
<tr>
<td>LISTCH</td>
<td>52</td>
<td>Current</td>
</tr>
<tr>
<td>LISTUN</td>
<td>53</td>
<td>List tape</td>
</tr>
<tr>
<td>LISTP</td>
<td>52, 53</td>
<td>For POGO restarts</td>
</tr>
<tr>
<td>RSTCLASS</td>
<td>54, 55</td>
<td>Restart Class</td>
</tr>
<tr>
<td>CHECK</td>
<td>56, 59</td>
<td>PSEUDO check sum</td>
</tr>
<tr>
<td>RSTPAR</td>
<td>50, 59</td>
<td>Restart parameters</td>
</tr>
</tbody>
</table>

The POGO INTERRUPT PROCEDURE (Chart 5) provides a standard method of communication with the operator and gives the operator the options of skipping or correcting calling cards. A message identifying the reason for entering the interrupt procedure is sometimes typed before entering it.
Block 01: Save the contents of location 0000, and plant a branch to the skip account routine. Type RSTOP.

Block 02: Halt for operator decision. To skip the current account, press program reset and start. This sends control to Block 14. Otherwise, press start and go to Block 03.

Block 03: Plant a branch to Block 06 at location 0000. Type RSTOP2.

Block 04: Halt for an operator decision. To correct the card, press program reset and start; control goes to Block 06; otherwise, press start and go to Block 05. The branch to Block 06 is made to provide for corrections to the current input record.

Block 05: No interrupt option has been specified, so restore location 0000, type IGNORMSG, and return to the point which called interrupt procedure. The subsequent action is up to the calling procedure.

Block 06: Correction option has been specified. Plant branch at location 0000 to Block 10. Type ICDMSG.

Block 07: Halt for operator decision. Press program reset and start to load a correction card from the console card reader. Otherwise, press start.

Block 08: Type TYPMSG to correct from typewriter. HP LABEL for such corrections.

Block 09: Restore location 0000 and return to Block 22, Chart 3.

Block 10: Restore location 0000 and read correction card from console card reader.
Block 11: If the card was in error, go to Block 13; if there was an end-of-file condition, go to Block 12; otherwise, go to Block 22, Chart 3.

Block 12: Type EOFMSG.

Block 13: Type ERRMSG.

Block 14: Restore location 0000 and turn off accounting switch. Go to get new input record.

Block 15: Halt to allow operator to reload card for reread.

Block 16: Halt to allow operator to reload card for reread.

Block 17: Get next input record.

The POGO END-OF-REEL PROCEDURE (Chart 6) is entered once for each tape mark encountered on the input tape. If entry is made with TMSW on (implying two successive tape marks), it will write four tape marks on each output tape and type ENDMSG. If start is pressed, both tapes will be rewound and unloaded and UNLOADMSG will be typed. If, instead of pressing start, a new card from console card reader is executed, output will continue to be stacked after the first TM.

Block 01: Set DTF for no rewind on file close.

Block 02: Close initial input file.

Block 03: Reset open switch so that subsequent entry to SYCA3 will cause input file to be opened.

Block 04: Is TMSW on?

Block 05: No, turn TMSW on.

Block 06: Go to TMRETURN, Chart 3, Block 13.

Block 07: Yes. Write four tape marks on appropriate tapes.
Block 08: Type ENDMSG.
Block 09: HP for operator decision.
Block 10: Type UNLOADMSG, unload input and output tapes.
Block 11: HB to normal restart.
Block 12: Control is turned to load button restart routine if operator pressed load button.

The POGO ACCOUNTING PROCEDURE (Chart 07) is used to account for the time used from the time a START ACCNT card is encountered until the time an END ACCNT is encountered. The END ACCNT causes an account card to be placed on the output to be punched by the 1401; it is also listed with the user's output so he has a record of the time he uses. Both types of card cause a message to be typed on the console typewriter. It contains the current account name and number, and the contents of the interval timer.

Block 01: Is the current input card an END ACCNT card? If it is go to Block 24, otherwise Block 02.

Block 02: Is it a START ACCNT?
Block 03: Is the POGO number of this input card _? This is the number inserted by the 1401 utility program.

Block 04: Reset the sequence number to zero.
Block 05: Is the ACCNTNO in the COMREC minus?
Block 06: Previous account is not yet closed (possibly a missing END ACCOUNT on that input deck), BLX to the end routine to force an END ACCNT. On return go to Block 08.

Block 07: Is current ACCNTNO zero?
Block 08: Scan for the account name and number. A name parameter is used for an account number and an alpha literal is used for an account name. The account number must consist of one letter followed by four digits. When valid account parameters are found they are placed in the COMREC.

Block 09: Has the scan of the operand been completed?

Block 10: Were both a valid account name and number found?

Block 11: Initialize console message.

Block 12: Is current card a start account card?

Block 13: Check the account parameters for invalid alpha combinations. A clobbered COMREC can cause them and these can cause tape write failures when putting out the account card.

Block 14: Initialize the output account card with the current account parameters.

Block 15: Is the MIXEDTAPE option in effect?

Block 16: Tape mark the LISTTAPE, write out the account card, write another tape mark.

Block 17: Reset the COMREC parameters for the closed account.

Block 18: Type the end-of-account message on the console typewriter.

Block 19: Tape mark the DECKTAPE, write out the account card, tape mark the DECKTAPE again.

Block 20: Type the start-of-account message on the console typewriter.

Block 21: Form CHECKSUM and plant in the COMREC. It is the sum of various, constant account parameters (constant for a given account, account name, number, etc.).
Block 22: Get the next input record.

Block 23: Return to the calling BLX + 1.

Block 24: BLX to the END routine to do the END ACCNT procedure.

After returning to get the next input record.

Block 25: Start of the END ACCNT procedure.

Block 26: Go to the OPTER error routine to insist upon a correction.

The RUNCHECK (Chart 8) routine in _SYCA3_ is used to put the appropriate code word on the output tapes during compilations and generator runs, to check the ready status of work tapes required, and to check the settings of the alteration switches.

Block 01: Is this a compile run control card?

Block 02: Is it a generator run control card?

Block 03: If the compile run is FORTRAN, go to Chart 11 to plant some SCANDAL codes in the COMREC.

Block 04: Set the ACCNTPARAM in the COMREC to show that the current account is an AUTOCODER compilation.

Block 05: Is the printed and punched output to be placed on the same tape?

Block 06: Place a tapemark and the appropriate 1401 code word on the LISTTAPE for no MIXEDTAPE option i.e. printed output only.

Block 07: Tapemark and place a code word on the LISTTAPE for the MIXEDTAPE option, punched and printed output.

Block 08: Place a tapemark and code word on the DECKTAPE for no MIXEDTAPE option, i.e. punched output only.
Block 09: The work tapes required for a compilation are checked for ready status. If there is a tape that is not ready, go to Block 10, otherwise go to Block 11. When the tape is ready, it is rewound.

Block 10: Type message to ready the specified tape.

Block 11: If all the alteration switches are off, return to Chart 3, Block 28.

Block 12: Type message to turn off any alteration switches that may be on.

Block 13: Halt for operator action, check again when start is pressed.

Arguments to the TM-TW routine (Chart 9) are presented to this routine through the accumulators. These include tape unit, and RDW when appropriate. This routine is used by the TAPE REMARKS function, the BLANK TAPE function and the POGO accounting procedure.

Block 01: Initialize tape commands and RDW.

Block 02: Check the ready status of tape unit. If ready, go to Block 04, otherwise go to Block 03.

Block 03: Type message to ready the specified tape.

Block 04: Execute operation wanted.

Block 05: Return to 0+CALL.

Block 06: Halt for operator action. Go to INTERPROCEDURE on program reset and start, go to Block 02 on Start.

The reading of POGO cards by .SYCA3 (Chart 10) can be switched from the systems standard input tape to the console card reader by the operator by altering CHANGENO (See COMREC page 16) to nines
Reading of POGO cards by the reader will continue until a POGO card with a blank opcode is reached.

Block 01: Read a POGO card from the console card reader.

Block 02: If there was a read error, go to Block 03, otherwise go to Block 04.

Block 03: Type error message to operator and wait for his action.

Block 04: Was there an end-of-file condition on the card read?

Block 05: Type a message to the operator and wait for his action.

Block 06: Is the opcode of the read POGO card blank?

Block 07: The blank opcode indicates the end of POGO cards in the console card reader. CHANGENO is reset to zero and .SYCA3 input is taken from INITAUNIT again.

There are certain operations in SCANDAL (Chart 11) determined by the parameters that are punched on the Compile Run control card. The options are LIST and FORCE. The presence of the word LIST in the operand will cause a listing of the source language cards to be made on the LISTTAPE along with the diagnostics. FORCE causes control to be turned over to FORTRAN even if there are diagnostics produced by SCANDAL. Since .SYCA3 and not SCANDAL reads the compile run card, it is necessary to present these options to SCANDAL via the COMREC. This routine analyzes the operand for these parameters.

Block 01: Set SCANLSTOPT to no list and SCANFRCOPT to no force.

Block 02: Scan the operand of the compile run control card.
Block 03: If an end-of-text return is given by SCAN, go to Block 04, otherwise go to Block 06.

Block 04: Tape mark and code word the LISTTAPE.

Block 05: Is this being run under the MIXEDTAPE option? If it is, return to Chart 8, Block 09, otherwise return to Chart 8, Block 08.

Block 06: Was the returned parameter a name? No, scan again.

Block 07: Was the name equal to FORCE?

Block 08: Was the name equal to LIST?

Block 09: Set SCANFRCOPT to force.

Block 10: Set SCANLSTOPT to list.

There is a routine in .SYCA3 belonging to POGO that checks alpha words for invalid alpha combinations. Arguments are presented to this routine one word at a time. If all characters are valid alpha characters, there is a normal return. If invalid alpha characters are found the location containing the argument will be changed to BAD. The only purpose of this routine is to check the account card before it is written on tape. Frequently run away loops cause information to be stored into cells of the COMREC that are usually alpha (the account name for example). If this were to be transmitted to tape, a tape write error would occur, so for convenience the account name and number are checked for invalid alpha combinations. The routine could easily be altered to check areas.
SCAN (Chart 12) is a routine used by the POGO functions in analyzing the operand of POGO calling cards. SCAN permits a key entry in context type of analysis of alphanumeric text.

SCAN is written as a translation matrix routine (although a degenerate form in size of control stack). The translation matrix will serve as a flow sheet for this routine. The various macros are listed below.

- **CLRL**: Clear cells used for packing names and integers.
- **SAVEX**: Save leading sign.
- **EN(X)**: Enter X into control pushdown.
- **SCANPP**: Process current pair.
- **CLRL**: Clear cells used for literals.
- **SCANPNP**: Process next pair.
- **EOR**: End of area to be scanned. Return to 0 + CALL.
- **MISS**: Missing parameter. Return to 1 + CALL.
- **BKSP**: Save current character for next scan.
- **PACKI**: Pack character into words.
- **TRANA**: Return to 2 + CALL with name right adjusted in accumulators 1 and 2. Accumulator 3 contains (right adjusted) the terminating character for the name in accumulators 1 and 2 (e.g. in "NAME," comma is the terminating character). The sign of accumulator 3 will be alpha unless an error in text is found, in which case the sign will be minus.
TRANL: Return to 3 + CALL with the RDW for the literal text area in accumulator 1. Accumulator 3 is treated as above in TRANA.

PACKL: Pack literal into words.

TRANI: Return to 4 + CALL with integer in accumulator 1. Accumulator 3 is treated as in TRANA.

To open SCAN execute the following orders:

```
ZA3 TEXTRDW
BLX SCANX, SCANINIT
```

To obtain the next entry from the operand execute the following orders:

```
BLX CALL, SCAN
B Returns
B End of SCAN area
B Missing parameter
B Name is found
B literal is found
B Integer is found
```

SCAN may be opened and called simultaneously by executing the following orders:

```
ZA3 TEXTRDW
BLX CALL, SCANOPEN
B
B
B
B
B
B
```
SCAN must be opened once for each text area that is to be scanned. Each subsequent call produces a return with the next sequential entity in the text area. The text RDW will normally be the operand of the input record.

This POGO OUTPUT ERROR routine (Chart 13) is typical of the error routine used in all sections of POGO. The number of attempts on a bad record is always 10 with the option for more.

Block 01: Mask priority and initialize the counter.

Block 02: If tape is in ready status, go to Block 04.

Block 03: Type message and halt. Press start (P.S.) to go to Block 02.

Program reset and start (PR, S) to go to OPTER.

Block 04: Execute tape operation.

Block 05: Wait for operation to finish.

Block 06: If the latch is not on, go to Block 15.

Block 07: If end-of-file, go to Block 10.

Block 08: If CNTR = 10, go to Block 13.

Block 09: Increment counter, backspace, skip. Go to Block 04.

Block 10: Type end-of-file message and halt PR, S to go to Block 11.

P.S. to go to Block 12.

Block 11: Tape mark, rewind and unload, Go to Block 04.

Block 12: Write four tape marks, rewind and unload. Go to Block 19.

Block 13: Type error message.

Block 14: Halt for operator decision. PR, S to OPTER.

Block 15: Continue on with the routine or function.

Block 16: Reset CNTR to zero.
Block 17: **OPTER** error routine.

Block 18: **OPTER** error routine.

Block 19: Halt for operator action.

**POGO MODIFICATIONS TO SYCL2**

**SYCL2**, including the POGO modifications, is loaded from the systems tape by the last section of the AUTOCODER processor. The normal conclusion of a single file compilation has been altered to never rewind the input and output tapes. The **COMREC** is then reloaded and a normal restart is given to **SYCA3** in place of **HALT 76**. The **LISTTAPE** is no longer rewound, but the **DECKTAPE** is following a MACRO generator run. The option of the halt after generation is to press start to go on with the systems run or program reset and start to give a neutral restart to **SYCA3**. The open and close routines are also included and the program compiled message has been shortened.

**POGO MODIFICATIONS TO UPDAT**

**UPDAT** (Chart 14) has been modified to permit the addition of a variable number of records of variable length in front of **BOOTSTRAP**. The maximum number of records is 12 with a total length of not more than 2500 instructions. After initial insertion, instructions in these records can be replaced on a one-to-one basis, but a change in length of record requires a revision of a systems tape that does not contain the records. The only modification to **UPDAT** has been made in the method of writing out **BOOTSTRAP**. The records are no longer moved to **NEWBLOCK**. The RDW's defining the records are loaded sequentially into the location used originally to define the writeout area.
Block 01: Load index TEMPREG with RDW's.

Block 02: Move a record RDW to NEWBOOT RDW.

Block 03: Is the next record the last record in BOOTSTRAP?

If so go to read in, but do not write record out.  
\textit{IOQRELSE} writes out last record.

Block 04: Go to IOCS to write out record.

Block 05: Is this the first run? (ALTSW 1 ON) If so go to Block 06.

Block 06: BLX around read routine to allow insertion of records in front of BOOTSTRAP.

Block 07: Is this the last record? If so go to \textit{IOQRELSE} to write this record out. Continue with normal \textit{UPDAT} functions.

\textbf{OTHER MODIFICATIONS}

The following coding blocks have been changed to no longer re-wind and unload the input tape at the conclusion of reading input:

\textit{SYCL5}  
\textit{SYCL6}  
\textit{SYCL7}

\textit{SYCL1} has been modified to allow for the loading of the debug routines.  
\textit{DEBUG} has been modified to dump on tape 23. \textit{FRT1B} has been modified to read its input from SCANDAL's output tape \textit{TAPESET} (2, 3). SCANDAL, a FORTRAN syntax check, has been added to the front of the FORTRAN COMPILER. MACROS. OP has been rewritten (Chart 29) and is used to generate linkage between FORTRAN programs and the POGO restart routines. It processes the FORTRAN STOP statement.
Block 01: The argument is transferred to a double digit restart parameter.

Block 02: Is this compilation for a 20K IBM 7074? If it is, go to Block 03; otherwise, go to Block 04.

Block 03: Generate the following:

```
ASSN
ZA1   ARG
STD1  9989(4, 5)+X1
B     9975+X1
```

Block 04: Generate the following:

```
ZA1   ARG
STD1  9989(4, 5)+X1
B     9975+X1
```

MACROPOGO (Chart 28) has been added to the system to close the loop back to POGO. It generates the linkage between AUTOCODER programs and the restart routines. The operand may be symbolic, blank, or contain missing parameters (indicated by "", ""). If the operand is blank or contains blank parameters, the corresponding entries in the POGO COMREC will be unchanged. If the operand contains symbolic parameters, they must be defined under a DA or a DC.

Block 02: Generate in line(BLX 94, POGO).

Block 03: Generate a NOP out of line.

Block 04: If object machine is in expanded mode, go to Block 07.

Block 05: Machine is not in expanded mode; turn additional storage switch off.
Block 06: Generate out of line (MSM 9989). Go to Block 09.

Block 07: Machine is in expanded mode; generate out of line (BASS*+4), (ASSN), (MSA 9989+X1), (B*+2), (MSP 9989+X1).

Block 08: Turn additional storage switch on.

Block 09: If there are any parameters, go to Block 15.

Block 10: No parameters; if the additional storage switch is on, go to Block 13.

Block 11: The additional storage switch not on; generate out of line B 9968.

Block 12: Leave generator.

Block 13: The additional storage switch is on; generate out of line (B9975+X1).

Block 14: Leave generator.

Block 15: There are parameters in the operand; add+12 to PCTR. 2.

Block 16: If first parameter is not blank, go to Block 19.

Block 17: First parameter is blank. If there are no more parameters, go to Block 26.

Block 18: There are no more parameters; move parameter zero plus index word PCTR. 2 to parameter one and go to Block 15.

Block 19: Parameter one is not blank; place field definition in WORK. 1 and WORK. 2.

Block 20: Is parameter one symbolic?

Block 21: Parameter one is symbolic; generate out of line (ZAI PAR 01).

Block 22: Is additional storage switch on?
Block 23: Additional storage switch is not on; generate out of line (STD1 9989) (WORK. 1, WORK. 2) and branch to Block 17.

Block 24: Additional storage switch is on; generate out of line (STD1 9989 (WORK. 1, WORK. 2+X1)) and branch to Block 17.

Block 25: Parameter one is not symbolic; generate out of line (ZAl TEXT .01).

Block 26: There are no parameters remaining. Is additional storage switch on?

Block 27: Additional storage switch is not on; generate out of line (B9968).

Block 28: Leave generator.

Block 29: Additional storage switch is on; generate out of line (B 9975+X1).

Block 30: Leave generator.

POGO FUNCTIONS

The simple POGO function REM (Chart 15) will be analyzed more as an example of how a function operates in the POGO framework.

All symbols are prefaced by REM except those referring to .SYCA3 or POGO.

Entrance to this function is made from POGO after it finds REM as an opcode in an accepted POGO calling card.

Block 01: Look for first word of LABEL in the REMTABLE.

Block 02: If not present go to Block 03, otherwise go to Block 04.

Block 03: Go to OPTER routine in POGO. OPTER will type invalid message and go to INTERPROC. INTERPROC will give the operator a choice of action in handling the error (see Chart 5).
Block 04: If **LABEL** is BLANK, go to Block 05. If **LABEL** is TYPE, go to Block 06. If **LABEL** is TAPE, go to Block 12.

Block 05: Go to START to get next input record. This is the return to POGO.

Block 06: Scan the operand for an alphanumeric literal.

Block 07: If literal is not found, go to OPTER ERROR ROUTINE, otherwise go to Block 08.

Block 08: Type REMRDW; it defines the area containing the found literal.

Block 09: Scan remainder of operand for HALT.

Block 10: If no HALT is found, go to Block 17.

Block 11: HP and wait for operator. START to go to Block 17.

Block 12: Scan operand for an alphameric literal.

Block 13: If no literal is found, go to Block 03, otherwise go to Block 14.

Block 14: Scan for channel and unit.

Block 15: If valid channel and unit found, go to Block 16, otherwise go to Block 03.

Block 16: Write literal on specified tape unit, and go to Block 17.

Block 17: Get next input record.

The ALTSW (Chart 16) function is used to type standard messages to the operator to set the alteration switches. It will then check again to see if he followed orders properly.

Block 01: Begin and initialize function.

Block 02: Scan for a parameter.

Block 03: If end-of-text return go to Block 04; otherwise go to Block 07.

Block 04: If a message was typed, go to Block 06, otherwise go to Block 05.
Block 05: Return to POGO.

Block 06: HB for operator action. Go to Block 01.

Block 07: If a name parameter was found, go to Block 08; otherwise go to Block 12.

Block 08: If name found was OFF, go to Block 10; otherwise go to Block 09.

Block 09: If name found was ON, go to Block 10; otherwise go to Block 02.

Block 10: Check switches according to OFF or ON.

Block 11: Type messages if necessary, go to Block 04.

Block 12: If an integer was found, go to Block 13; otherwise go to Block 02.

Block 13: If the integer is valid, go to Block 14, otherwise Block 21.

Block 14: Initialize for individual switch settings.

Block 15: Scan for a parameter.

Block 16: If end-of-text, go to Block 21; otherwise go to Block 17.

Block 17: If found OFF, go to Block 18; otherwise go to Block 20.

Block 18: If specified switch is set properly, go to Block 02; otherwise go to Block 19.

Block 19: Type message to set switch, go to Block 02.

Block 20: If found ON, go to Block 18; otherwise go to Block 15.

Block 21: Go to OPTER error routine to insist upon a correction.

The POGO LOAD FUNCTION (Chart 17) is used to load AUTOCODER programs. It initializes the core as specified by the programer and dumps the 5/ card loader initialization instructions into locations zero and upward. It will also make a SNAPSHOT routine available.
Block 01: Initialize switches, index words, etc. Scan for a name parameter.

Block 02: Is the found parameter a name?

Block 03: Is the name TAPE?

Block 04: The channel and unit where the program is to be loaded from is picked up in a scan. If none is found or if the found one is invalid control goes to INTERPROC. The initialization instructions for the 5/ card loader are then initialized with the found channel and unit.

Block 05: The initialization instructions are moved to zero.

Block 06: Scan for the next parameters. At this point all the required parameters have been found. The rest are optional.

Block 07: What is the parameter type of the found parameter?

Block 08: Use the found integer to set the current bank of electronics switches.

Block 09: Is the found name SNAPSHOT?

Block 10: An RDW defining the area available to the SNAPSHOT routine is obtained by scanning for two integers that are used as start and stop addresses of the area. A switch is set to indicate that SNAPSHOT is to be used.

Block 11: Is the name HIGHDENSITY? This is used to specify the density of the tape being loaded.

Block 12: Set the tape in high density.

Block 13: Is the name LOWDENSITY?
Block 14: If the tape unit is not a tape that the system expects to be in high density, set it in low density. Otherwise type a message and provide normal error options.

Block 15: Is name equal to cards?

Block 16: The synchronizer is picked up by scanning for an integer. If it is valid the initialization instructions are set for this synchronizer, otherwise the normal error options are provided.

Block 17: Is a SNAPSHOT called for?

Block 18: Is LABEL zero?

Block 19: Put out a message instructing the operator to press one of the control keys. On computer reset and program reset, his actions are checked.

Block 20: Is the core to be zeroed?

Block 21: Zero upper 10K if an addstorage program, move the lower core zero routine to area of execution.

Block 22: Is the core to be restored?

Block 23: Initialize the core restore routine.

Block 24: Move the restore routine to its area of execution.

Block 25: Modify .SYCA3 so it can be used to read control cards.

Block 26: Construct a parameter area from the control cards.

Block 27: Modify exits from LOAD into the 5/card loader initialization instructions to exit into the SNAPSHOT loader.

Block 28: Move parameter area and SNAPSHOT loader to area of executions.
Block 29: Control goes to zero routine.

Block 30: Control goes directly to loader. SYCA3 is overwritten.

Block 31: Control is turned over to the restore routine.

Block 32: Halt for the operator.

BLANK TAPE (Chart 18) is used to execute special tape orders; that is the one indicated by position nine of tape orders having position five equal to zero.

Block 01: Look up LABEL in TAPETABLE.

Block 02: If found go to Block 04.

Block 03: Go to OPTER.

Block 04: Go to the appropriate tape label routine. This may involve loading another block from the systems tape. BLANK tape and COPY-DUMP tape are one block and POSITION and SEARCH are in the second.

Block 05: Tape label is blank, i.e. BLANKTAPE.

Block 06: Plant PR, S branch at 0000.

Block 07: Scan for name.

Block 08: If found a name, go to Block 10. If not an end-of-text, go to Block 07.

Block 09: Return to POGO.

Block 10: Look up name in TAPEOPTAB.

Block 11: If not found, go to Block 07.

Block 12: Initialize tape order and messages.

Block 13: Scan for the parameter.

Block 14: If EOR return, go to Block 09. If integer found, go to Block 15; otherwise go to Block 25.
Block 15: If channel and unit is valid, go to Block 17.

Block 16: Go to OPTER.

Block 17: Initialize for channel and unit.

Block 18: If channel and unit is a special tape, go to Block 19.

Block 19: Type warning message and halt.

Block 20: If PR, S go to interrupt routine.

Block 21: If tape is ready, go to Block 23.

Block 22: Type warning message and halt. PR, S to interrupt routine, start to Block 21.

Block 23: Execute tape order.

Block 24: Type message, go to Block 13.

Block 25: Is the parameter a name?

The COPY-DUMP TAPE function (Chart 19) is used to copy information from one tape to another or to break large records down into 10 word records so they may be printed by POGOINOUT.

Block 01: If the LABEL is COPY, go to Block 02.

Block 02: Initialize for COPY. Go to Block 04.

Block 03: Initialize for DUMP.

Block 04: Scan for and initialize tape units. Also initialize priority.

Block 05: Scan for and initialize the number of records, segments, or files. Maximum number is 999.

Block 06: Read a record.

Block 07: If not a segment mark, go to Block 11.

Block 08: If DUMP, go to Block 10.

Block 09: Write segment mark. Go to Block 19.
Block 10: Write segment mark message, go to Block 19.
Block 11: If not tape mark, go to Block 15.
Block 12: If DUMP, go to Block 14.
Block 13: Write tape mark. Go to Block 19.
Block 14: Write tape mark message. Go to Block 19.
Block 15: Set correct record length.
Block 16: If COPY, go to Block 18.
Block 17: Dump record into 10-word records. Go to Block 19.
Block 18: Copy record.
Block 19: If COUNTSW off, go to Block 06.
Block 20: Increment counter.
Block 21: If done, go to Block 22; otherwise go to Block 06.
Block 22: Get next input record.

The POSITION TAPE function (Chart 20) is used to position a tape N records, segments, or tape marks forward or backward.
Block 01: Initialize switches and program.
Block 02: Scan and initialize tapes.
Block 03: Scan and initialize for type of tape element and the number of elements.
Block 04: Scan for and initialize direction.
Block 05: If positioning by tape marks, go to Block 16.
Block 06: If positioning by segment marks, go to Block 12.
Block 07: If forward, go to Block 09.
Block 08: Backspace go to Block 10.
Block 09: Read a record.
Block 10: Increment the counter.

Block 11: If this is not the last, go to Block 07; otherwise go to Block 24.

Block 12: If FORWARD, go to Block 14.

Block 13: Tape segment backspace N segments go to Block 15.

Block 14: Tape segment forward N segments.

Block 15: Go to START.

Block 16: If FORWARD, go to Block 23.

Block 17: Backspace.

Block 18: Read a record.

Block 19: If it is a tapemark, go to Block 21.

Block 20: Backspace, go to Block 17.

Block 21: Increment the counter.

Block 22: If last, go to Block 25; otherwise go to Block 20.

Block 23: Read a record.

Block 24: Go to START to get next input record.

Block 25: Go to START to get next input record.

Block 26: Go to START to get next input record.

Block 27: Was it a tapemark?

Block 28: Increment the counter.

Block 29: Last record.

The POGO SEARCH TAPE FUNCTION (Chart 21) is used to search tapes for certain records.

Block 01: Initialize switches index words, etc.

Block 02: Scan for the tape unit containing the tape to be searched.
Block 03: Was an integer found?
Block 04: Scan for the search argument.
Block 05: Is an argument available?
Block 06: Save the argument.
Block 07: Scan for BACKSPACE, SEGMENT, or the position of the argument in the record.
Block 08: What kind of parameter was returned?
Block 09: Operand scan completed?
Block 10: Was a name returned?
Block 11: Is the name BACKSPACE?
Block 12: Is the name SEGMENT?
Block 13: Set the switch in the EOS routine so that the argument is expected to be in a record following a segment work.
Block 14: Set the switch in the EOJ routine so the found record is backspaced over.
Block 15: The found integer sets the position of the search argument in the record.
Block 16: Initialize tape commands with specified CU.
Block 17: Read a record from the search tape.
Block 18: Is read operation completed?
Block 19: Was a tape mark read?
Block 20: Is the I/O option being used? This is used primarily for searching POGO input and output tapes. EOF conditions, with this option, occur only after reading two successive tape marks on an input file, four on output.
Block 21: Have the maximum number been read?
Block 22: Provide the first set of end-of-file options.
Block 23: Halt for operator decision.
Block 24: Was a segment mark read?
Block 25: Was the read in error?
Block 26: Have there been 10 consecutive errors?
Block 27: Are we ignoring errors?
Block 28: Provide error options.
Block 29: Halt for operator decision.
Block 30: Backspace over record.
Block 31: Is this the record being searched for?
Block 32: Is the segment procedure being used?
Block 33: Skip forward one segment.
Block 34: Is record to be backspaced over?
Block 35: Backspace over found record.
Block 36: Type completion message.
Block 37: Exit back to POGO for next input record.
Block 38: Provide second set of end-of-file options. Halt for operator decision.
Block 39: Rewind the search tape and continue searching.

The DUMP (Chart 22) function is used to obtain core dumps. A mnemonic or constant dump is available. The dump had two phases; the first phase analyzes the operand and is discussed below.

Block 01: Initialize all program switches.
Block 02: If \texttt{LABEL} is CONSTANT, go to Block 04; otherwise go to

Block 03.
Block 03: Locate mnemonic dump phase, go to Block 05.

Block 04: Locate constant dump phase.

Block 05: Set carriage control.

Block 06: If dump tape is specified, go to Block 08.

Block 07: Force dump to LISTTAPE. Go to Block 12.

Block 08: If tape channel and unit is valid, go to Block 12.

Block 09: Plant branch to OPTER at zero.

Block 10: Type message and halt.

Block 11: If PR, S, go to OPTER, start to continue.

Block 12: Scan for integer pairs.

Block 13: If not end-of-scan, go to Block 15.

Block 14: Wrap up RDW analysis; if there is one invalid RDW the operator is given the option of continuing with the dump or ignoring it. The dump is discontinued if NO valid RDW's are present and he elects to continue.

Block 16: If start address is invalid, go to Block 23.

Block 17: PROCESS, go to Block 21.

Block 18: If stop address is invalid, go to Block 19.

Block 19: If stop address is less than start address, go to Block 21.

Block 20: PROCESS.

Block 21: If RDW is invalid, go to Block 23.

Block 22: PROCESS as valid.

Block 23: Type error message.

Block 24: Initialize for next RDW, go to Block 12.

Block 25: Initialize phase two loader.
Block 26: Initialize core restore and core save.

Block 27: Move routines to area of execution.

Block 28: Execute restore.

Block 29: Go to OPTER to insist upon correction.

The CONSTANT DUMP (Chart 23) may be called by the first phase of the DUMP function. It is loaded into locations 176-300 and gives a neutral restart when completed.

Block 01: Initialize switch settings, index registers, tape commands, etc.

Block 02: Get an RDW from the RDW list planted by phase 1.

Block 03: Has the previous RDW been exhausted? If it has, go to Block 15; otherwise go to Block 04.

Block 04: Initialize index registers for a new line of output.

Block 05: If the current line is all zeros, go to Block 03 and forget about it; otherwise go to Block 06.

Block 06: If the current cell of output is alpha, go to Block 07; otherwise go to Block 10.

Block 07: Is the current alpha cell a record mark? If it is, go to Block 09; otherwise go to Block 08.

Block 08: Plant the alpha word in the output area, go to Block 11.

Block 09: Plant the alpha words RECORD MARK in the output area to replace the record mark, go to Block 11.

Block 10: Edit current cell into the output area.

Block 11: If this cell is the last one of the current line, go to Block 12; otherwise go to Block 06.
Block 12: Remain in a small waiting loop until the previous line has been written on tape.

Block 13: The current line is written on tape, the area to contain the next line is zeroed.

Block 14: If the next RDW is needed, go to Block 15; otherwise go to Block 04.

Block 15: Are there RDW's remaining in the list? If there are, go to Block 02; otherwise go to Block 16.

Block 16: Put the last record on tape, type a completion message to the operator, and return to POGO.

The POGO FUNCTION MNEMONIC DUMP (Chart 24) is used to obtain a core dump, it is called by the first phase of the DUMP function. A linkage routine between phase one and two saves the core used by phase two on a tape. After this is done phase two is loaded and executed.

The output consists of five core locations per line. Each location has its corresponding mnemonic opcode followed by the sign and ten digits of that location printed out.

Block 01: Initialize tape operations, index words, RDWs and switches.

Block 02: Is the current output RDW negative? If it is the last RDW of the current block has been reached.

Block 03: Make the RDW plus, set the block out switch on.

Block 04: Is the current dump RDW (those specified by user) exhausted?

Block 05: Is another dump RDW available?

Block 06: Is current dump RDW minus?

Block 07: Position DECKTP in front of the record to be restored.
Block 08: Restore the area indicated by current RDW. The area to be restored is read into core, but not into the actual locations it previously occupied.

Block 09: Adjust RDW to refer to the relocated section of core.

Block 10: Initialize the current RDW for the output block.

Block 11: Is the current five locations of output all zero?

Block 12: The current output location is edited into the output area.

Block 13: Get the mnemonic opcode from a table by using the first two digits of the core location as an index.

Block 14: Is the retrieved value numeric?

Block 15: Plant the mnemonic opcode in the output area.

Block 16: Has the last cell of the current line been processed?

Block 17: Increment index registers, etc. for next output location.

Block 19: Has one of the output blocks been filled?

Block 20: Get on output RDW from the output list. It defines one record in the block.

Block 21: Plant record marks at end of each record in the block.

Block 22: Is previous block out?

Block 23: Write out current block.

Block 24: Zero next block.

Block 25: Initialize RDWS and index words for the current block.

Block 26: Put out the last block.

Block 27: Position the DECKTP in front of the first SNEAK record.

Block 28: Type completion message on console.

Block 29: Return to POGO with a neutral restart.
The POGO LIBRARY function (Chart 25) is used to add programs to the library tape. The program to be added must not occupy locations 180-299. The program will be loaded by the five/card loader with an execute card to 180 following the program. The bounds of the program and the entry point are contained on the POGO card.

Block 01: Initialize switches and tape.

Block 02: If no X is in column 15, go to Block 04.

Block 03: Turn on XSW.

Block 04: Scan for integer.

Block 05: If not end of text, go to Block 07.

Block 06: Go to OPTER, insist upon correction.

Block 07: If not found, go to Block 04.

Block 08: Initialize input tape.

Block 09: Scan for literal.

Block 10: If not end of text, go to Block 12.

Block 11: Go to OPTER, insist upon correction.

Block 12: If not found, go to Block 09.

Block 13: Scan and build RDW's and the entry point.

Block 14: Initialize the second phase.

Block 15: Zero core and initialize for the appropriate loader.

Block 16: Read a record from the library tape.

Block 17: If not end-of-tape, go to Block 16.

Block 18: Add new program to tape.

Block 19: Type the id record.

Block 20: Give a neutral restart.
The POGO LLOAD (Chart 26) function is used to load programs from the library tape. Electronic switch banks will be initialized if integers are present in the operand. If a Z occurs in col. 14, the core will be zeroed first. The first phase analyzes parameters and sets up the call for the second phase. The second phase will be left in core to be used by the loaded routine. By entering the routine at 180, the loaded routine will be executed. Entering at 182 will return control to calling routine after loading. Preceding each ID record is a segment mark. Following each ID record is a program record.

Block 01: Initialize Scan and scan for a literal.
Block 02: If not end of text, go to Block 04.
Block 03: Go to OPTER.
Block 04: If not found go to Block 01.
Block 05: Initialize search argument.
Block 06: Load phase 2 block.
Block 07: Scan for an integer.
Block 08: If not end-of-text, go to Block 12.
Block 09: If zeroing is not specified, go to Block 11.
Block 10: Zero the core except the locations that transfer the argument to phase 2 and the area occupied by phase 2.
Block 11: Go to Block 14, phase 2.
Block 12: If not found, go to Block 07.
Block 13: Initialize the current electronic switch and increment the counter. Go to Block 07.
Block 14: Initialize for execute. Go to Block 16.
Block 15: Initialize for no execute.
Block 16: Initialize search, etc.
Block 17: Segment space forward.
Block 18: Read the ID record.
Block 19: If this is not the first record, go to Block 21.
Block 20: Save this name to terminate the search.
Block 21: If this is not the last, go to Block 27.
Block 22: If not execute, go to Block 24.
Block 23: Execute branch in ID record.
Block 24: If not called from LLOAD, go to Block 26.
Block 25: Return to LLOAD, type message and give neutral restart.
Block 26: Give current restart.
Block 27: Found.
Block 28: Executive active?
Block 29: Execute loaded program.
Block 30: Return.

The POGO FORT FUNCTION (Chart 27) provides the loader and package and IBM library functions at object time. These are pulled from the systems tape. A SNAPSHOT can also be provided.
Block 01: Locate the FORTRAN loader and package on the systems tape.
Block 02: Scan the operand of the card for the names of the functions to be loaded from the systems tape.
Block 03: Is the found parameter a name?
Block 04: Is the name SNAPSHOT?
Block 05: Store the name in the FORTGET table.
Block 06: Check the FORTNEEDS table to see if this function requires another one.

Block 07: Is another required? ex: SIN-COS.

Block 08: Set the SNAPSHOT switch to yes.

Block 09: Is the scan completed?

Block 10: Sort the FORTGET table into the FORTAB.

Block 11: Load the package and loader.

Block 12: Is SNAPSHOT to be loaded?

Block 13: Build a parameter area from the parameter cards loaded behind the FORT card.

Block 14: Move the parameter area and the SNAPSHOT loader to the area of execution.

Block 15: Zero core up to the bottom of the program (FORT).

Block 16: Search for the current function.

Block 17: Is current function on systems tape?

Block 18: Load and relocate the program, step to next one.

Block 19: Is SNAPSHOT to be loaded?

Block 20: Locate SNAPSHOT. Set COMREC code so the loader knows SNAPSHOT is to be loaded.

Block 21: Zero core up to FORTRAN loader's table.

Block 22: Transfer control to FORTRAN loader.

**ADDITION OF FUNCTIONS**

Additional functions may be added to POGO very easily by means of a SYSTEMS RUN. If the function is to be a new coding block, it takes five patch cards to `SYCA3` and the new block to be entered. The
OPTABLE needs two additional entries. The first is the alpha opcode-name of the function; the second contains in positions two to five the address of the call to PROCESSOR2 for that function. The RDW defining the table needs to be incremented to allow for this new entry. The call to PROCESSOR2 requires two patch cards and is placed in the list at the end of the REM function. This entry is as follows:

```
BLX M1, PROCESSOR2
DC
POGOXXXXX
```

"XXXXX" is the name of the new function.

If the new function is to be a patch onto .SYCA3, care must be taken to see that it isn't overwritten by the loading of other functions. Except for very small functions, reassembly of POGO would be required for this type of addition in order to change the ORIGIN CONTROL for the functions. This type of addition requires three patches to .SYCA3. These are the ones concerned with the patching of the OPTABLE which was mentioned previously. The address in this case, however, is now the entry point of the function.

Functions may be patched onto other coding blocks. This requires that the coding block being patched be patched so that it is aware of the presence of the new addition. Care must also be taken to assure that the area being patched does not overwrite part of the patched block, or is not being used by it as an input/output area, etc.
There are three routines in .SYCL1 which are very useful in writing new POGO functions. These are STCLOCATE, STCLOAD, and STCPERFORM. If the new function is a new coding block it may be located, loaded, and performed by use of these. STCLOCATE is a priority search that allows overlapping of search time and operand analysis if necessary. In order to use these in an expanded machine, the memory must be in the unexpanded mode. A POGO routine EXPCLOSE may be used to do this.

The following orders will locate, load, and execute the systems tape block named in the DC below the transfer to STCLOCATE:

BLX CALL, EXPCLOSE
BLX IOQRETURN, STCLOCATE For locating block
DC . @10CHARNAME@
BLX IOQRETURN, STCLOAD For loading block
 BLX IOQRETURN, STCBRANCH To execute block

Alternatively if a locate, load, and execute is needed the following orders are sufficient:

BLX CALL, EXPCLOSE
BLX M1, PROCESSOR2
DC @10CHARNAME@
To load and execute a coding block, the following orders are sufficient. These need to be preceded by a locate.

BLX CALL, EXPCLOSE
BLX M1, STCPERFORM

MAINTENANCE OF THE SYSTEM

Maintenance is required to update the system every time an IBM version or level change occurs. Since POGO is attached to the compiler system in only four places, it is generally very easy to update from version to version. The only coding blocks patched in POGO are .SYCA3, .SYCL2, .SYCL1, and UPDAT. These coding blocks should be checked to see if any important changes or patches have been made which would interfere with POGO. If any conflict occurs, POGO will need to be reassembled.

POGINOUT 1401 PROGRAM

The POGINOUT program for an IBM 1401 Tape Systems performs Card-to-Tape (with printed log), Tape-to-Card and Tape-to-Printer operations. The functions to be performed are specified by the user by a series of code-word tape records or sense switches, which are located on the 1401 console. The 1401 system necessary to operate this program includes:

1. 1401 Model C3 (Tape system with 4000 storage positions)
2. High-Low-Equal Compare feature
3. Advanced Programming feature
4. 1402 Card Read Punch Unit
5. 1403 Printer, Model 2
6. 729 Magnetic Tape Units, Model II or IV
7. Six additional sense switches.

For further information on the POGINOUT program, see the 1401 Multiple Utility POGINOUT Manual.

Acknowledgments

The POGO system owes a great deal to the flexibility of the IBM 7070 Compiler System and is an extension of it. M. K. Rhyne and W. J. Higby helped check out and debug the system.
01
**UNEXPANDED MODE**
**RESTART**

02
**EXPANDED MODE**
**RESTART**

03
**SET PRIORITY**
**MASK TO ALLOW NO INTERRUPTS**

04
**TURN ON**
**ADDITIONAL STORAGE SWITCH**

05
**MAKE SIG***
**RESTART**

06
**SET PRIORITY**
**MASK TO ALLOW NO INTERRUPTS**
**MSP REPAIR**

07
**WAIT FOR CHANNEL TO BECOME FREE**

08
**THE 15**
**THE 15**

09
**SAVE INDEX WORDS**
**0-14**

10
**READ IN FIRST PHASE OF SNEAK**

11
**PLANT**
**RESTART PARAMETERS**

12
**WAIT FOR SNEAK**
**TO BE READ**

13
**EXECUTE SNEAK**
START

D1
SET OPTIONS TO
NO

D2
PARAMETERS

END
YES

N0
NAME
PARAMETER

N0
NAME
EQUAL
FORCE

N0
NAME
EQUAL
LIST

END
YES

TH-CODE-WORD
THE LIST TAPE

YES
SETTAPE
OPTION

NO
SET LIFT
OPTION

CONNECT TO YES

NAME
EQUAL
LIST

NAME
EQUAL
FORCE

NAME
PARAMETER
<table>
<thead>
<tr>
<th>Incoming Character</th>
<th>Original</th>
<th>Alpha-</th>
<th>Literal</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ DR -</td>
<td>CLR I</td>
<td>BKSP</td>
<td>PACK L</td>
<td>BKSP</td>
</tr>
<tr>
<td></td>
<td>SAVEX</td>
<td>TRA A</td>
<td></td>
<td>TRA N</td>
</tr>
<tr>
<td></td>
<td>ENV(I)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Digit</td>
<td>CLR I</td>
<td>PACK I</td>
<td>PACK L</td>
<td>PACK I</td>
</tr>
<tr>
<td></td>
<td>ENV(A)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Letter</td>
<td>CLR I</td>
<td>PACK I</td>
<td>PACK L</td>
<td>BKSP</td>
</tr>
<tr>
<td></td>
<td>ENV(A)</td>
<td></td>
<td></td>
<td>TRA N</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Period (.)</td>
<td>CLR I</td>
<td>PACK I</td>
<td>PACK L</td>
<td>BKSP</td>
</tr>
<tr>
<td></td>
<td>ENV(A)</td>
<td></td>
<td></td>
<td>TRA N</td>
</tr>
<tr>
<td></td>
<td>PP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alpha</td>
<td>CLR L</td>
<td>BKSP</td>
<td>TRAN L</td>
<td>BKSP</td>
</tr>
<tr>
<td></td>
<td>ENV(L)</td>
<td>TRA N</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Blank</td>
<td>PNP</td>
<td>TRA N</td>
<td>PACK L</td>
<td>TRA N</td>
</tr>
<tr>
<td>EOR</td>
<td>EOR</td>
<td>TRA N</td>
<td>TRAN L</td>
<td>TRA N</td>
</tr>
<tr>
<td>Otherwise</td>
<td>MISS</td>
<td>TRA N</td>
<td>PACK L</td>
<td>TRA N</td>
</tr>
</tbody>
</table>

SCAN is a subroutine used by POGO functions to analyze operands of calling cards.
01
LOAD TEMPRG.

02
MOVE RGN TO REBOOT.

03
NEXT
RECORD LAST
BOOTSTRAP.
YES
NO
UPDAT READ LOOP

04
LET IOS WRITE
OUT THE
RECORD.

05
ALTSW
ONE ON
YES
NO
IDRS READ
ROUTINE

06
SIX AROUND
READ ROUTINE.

07
IDRS READ
ROUTINE

08
LAST
RECORD
YES

09
NO
IDMRELEASE
START

01

DO NOT PACKAGE AND LOAD IN THE SYSTEM'S TAPE.

02

READ THE OPERAND OF THE CARD FOR REQUIRED PAR.

03

END OF TEXT NO

04

PARAMETER NO

05

SNAPSHOT NO

06

STORE NAME IN FORTGET TABLE

07

DOES THIS REQUIRE ANOTHER NO

08

SET SNAPSHOT SWITCH 13

09

CHECK FUNCTION AGAINST FORT NEEDS TABLE.

10

YES

11

SORT FORTGET INTO FORTAB.

12

LOAD THE PACKAGE AND LOADER.

13

DISPLAY CUBE OF FORGET

14

MOVE PAR. AREA AND SNAP LOADER TO AREA OF EXECUTION.

15

READ CORE UPTO BOTTOM OF FORFI.

16

SEARCH FOR CURRENT FUNCTION.

17

LOAD AND RECYCLE IF NEXT STEP TO NEXT FUNCTION.

18

LOAD AND RECYCLE IF NEXT STEP TO NEXT FUNCTION.

19

CHECK CORE TO SEQUENCE CORRECT.

20

LOCATE THE SNAPSHOT SET SEQUENCE CORRECT.

21

SEND REST OF CORE.

22

GO TO FORTRAN.