2000C HIGH SPEED
TIME SHARED BASIC
INTERNAL MAINTENANCE SPECIFICATIONS

BILL HACCOU
RICH PEARSON

August 10, 1972
SYSTEM LIBRARY ROUTINES

ASSIGN------------------------------------------71
CHAIN------------------------------------------72
SAVE------------------------------------------74
CSAVE------------------------------------------75
SUPERSAVE--------------------------------------76
GET--------------------------------------------80
APPEND----------------------------------------81
HELLO------------------------------------------82
BYE-------------------------------------------83
KILL-------------------------------------------84
RENUMBER---------------------------------------85
NAME------------------------------------------88
CATALOG----------------------------------------89
LIBRARY----------------------------------------90
GROUP------------------------------------------90
DIRECTORY - USER CONSOLE------------------------91
SDIRECTORY - USER CONSOLE-----------------------92
REPORT - USER CONSOLE---------------------------93
STATUS - USER CONSOLE---------------------------94
DELETE-----------------------------------------95
TIME------------------------------------------96
PROTECT----------------------------------------97
UNPROTECT--------------------------------------98
OPEN------------------------------------------99
LENGTH-----------------------------------------101
ECHO------------------------------------------102
MESSAGE----------------------------------------103
L PRINTER--------------------------------------103-A
REPORT - SYSTEM CONSOLE------------------------104
DIRECTORY - SYSTEM CONSOLE----------------------105
SDIRECTORY - SYSTEM CONSOLE--------------------106
STATUS - SYSTEM CONSOLE------------------------107
ROSTER----------------------------------------108
ANNOUNCE---------------------------------------109
RESET-----------------------------------------110
CHANGEID---------------------------------------111
SLEEP-----------------------------------------112
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hibernate</td>
<td>113</td>
</tr>
<tr>
<td>NEWID</td>
<td>114</td>
</tr>
<tr>
<td>KILLID</td>
<td>116</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>117</td>
</tr>
<tr>
<td>LOCK</td>
<td>118</td>
</tr>
<tr>
<td>MUNLOCK</td>
<td>119</td>
</tr>
<tr>
<td>MLOCK</td>
<td>120</td>
</tr>
<tr>
<td>COPY</td>
<td>122</td>
</tr>
<tr>
<td>BESTOW</td>
<td>123</td>
</tr>
<tr>
<td>SANCTIFY</td>
<td>124</td>
</tr>
<tr>
<td>DESECRATE</td>
<td>125</td>
</tr>
<tr>
<td>PURGE</td>
<td>126</td>
</tr>
<tr>
<td>MAGTAPE</td>
<td>128</td>
</tr>
<tr>
<td>PHONES</td>
<td>129</td>
</tr>
<tr>
<td>PRINTER</td>
<td>130</td>
</tr>
<tr>
<td>SPEED</td>
<td>131</td>
</tr>
<tr>
<td>PORT</td>
<td>132</td>
</tr>
<tr>
<td>1/O Processor Program</td>
<td>133</td>
</tr>
<tr>
<td>2100A Asynchronous Channel Multiplexor</td>
<td>133</td>
</tr>
<tr>
<td>Send Channel Parameter</td>
<td>133</td>
</tr>
<tr>
<td>Receive Channel Parameter</td>
<td>133</td>
</tr>
<tr>
<td>MULTIPLEXER DRIVER</td>
<td>134</td>
</tr>
<tr>
<td>Initialization</td>
<td>134</td>
</tr>
<tr>
<td>Receive Channel Processing</td>
<td>134</td>
</tr>
<tr>
<td>Send Channel Processing</td>
<td>134-A</td>
</tr>
<tr>
<td>Abort Processing</td>
<td>134-A</td>
</tr>
<tr>
<td>Multiplexer End of Processing</td>
<td>134-A</td>
</tr>
<tr>
<td>LINE PRINTER DRIVER</td>
<td>135</td>
</tr>
<tr>
<td>2100 DATA SET CONTROL INTERFACE</td>
<td>136</td>
</tr>
<tr>
<td>DATA SET CONTROL BOARD DRIVER</td>
<td>137</td>
</tr>
<tr>
<td>INITIALIZATION</td>
<td>138</td>
</tr>
<tr>
<td>POWER FAIR &amp; RECOVERY</td>
<td>139</td>
</tr>
<tr>
<td>TIME BASE GENERATOR</td>
<td>140</td>
</tr>
<tr>
<td>TELETYPE TABLES</td>
<td>141</td>
</tr>
<tr>
<td>SELECTRIC CONVERSION ROUTINES</td>
<td>143</td>
</tr>
<tr>
<td>CONVERSION TABLES FOR THE IBM 2741 TERMINALS</td>
<td>145</td>
</tr>
<tr>
<td>TRANSMITTING EBCD &amp; CALL/360 CODES</td>
<td></td>
</tr>
<tr>
<td>HARDWARE CONFIGURATION</td>
<td>146</td>
</tr>
</tbody>
</table>

iii
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESSOR INTERCONNECT</td>
<td>147</td>
</tr>
<tr>
<td>TWO PROCESSOR POWER FAIL CHARACTERISTICS</td>
<td>150</td>
</tr>
<tr>
<td>CORE ALLOCATION IN I/O PROCESSOR PROGRAM</td>
<td>153</td>
</tr>
<tr>
<td>BASIC</td>
<td></td>
</tr>
<tr>
<td>SYNTAX</td>
<td>162</td>
</tr>
<tr>
<td>COMPILATION</td>
<td>162</td>
</tr>
<tr>
<td>VALUE</td>
<td>163</td>
</tr>
<tr>
<td>DECOMPILATION</td>
<td>164</td>
</tr>
<tr>
<td>PRNST</td>
<td>164</td>
</tr>
<tr>
<td>EXECUTION</td>
<td>164</td>
</tr>
<tr>
<td>FORMATTER</td>
<td>174</td>
</tr>
<tr>
<td>ERROR ROUTINES</td>
<td>177</td>
</tr>
<tr>
<td>CORE MAPS</td>
<td>178</td>
</tr>
<tr>
<td>INTERNAL REPRESENTATION</td>
<td>181</td>
</tr>
<tr>
<td>VARIABLE STORAGE ALLOCATION</td>
<td>186</td>
</tr>
<tr>
<td>FILE TABLE ENTRY</td>
<td>188</td>
</tr>
<tr>
<td>FILE CONTENTS</td>
<td>189</td>
</tr>
<tr>
<td>UPDATE LAST CHANGE DATE ROUTINE</td>
<td>190</td>
</tr>
<tr>
<td>RUN-TIME STACKS</td>
<td>191</td>
</tr>
<tr>
<td>LANGUAGE PROCESSOR TABLES</td>
<td>193</td>
</tr>
<tr>
<td>BASIC FLOWCHARTS</td>
<td>194</td>
</tr>
<tr>
<td>LOADER</td>
<td>246</td>
</tr>
<tr>
<td>LOADER SUBROUTINES</td>
<td>250</td>
</tr>
<tr>
<td>LOADER FLOWCHARTS</td>
<td>256</td>
</tr>
<tr>
<td>BOOTSTRAP PROCEDURE</td>
<td>320</td>
</tr>
<tr>
<td>SLEEP TAPE FORMAT</td>
<td>321</td>
</tr>
</tbody>
</table>
INTRODUCTION

The 2000C (HIGH SPEED) TIME SHARED BASIC SYSTEM consists of three separate programs which are run on two processors. The Communications processor Program is responsible for handling all multiplexed I/O from user terminals. The System contains the BASIC interpreter, executive and library routines and runs on the main processor. The Loader, which also runs on the main processor, is responsible for generating initial systems, backing up the system on mag tape, reloading the entire system and user library, and selectively reloading or backing up users libraries.

HARDWARE CONFIGURATION

1) UP TO 16 TERMINALS

10 PROCESSOR INTERCONNECT
11 PROCESSOR INTERCONNECT
12 TIME BASE GENERATOR
13 1st MULTIPLEXOR
14 1st MULTIPLEXOR
15 DATA SET CONTROL FOR 1st MULTIPLEXOR
16 LINE PRINTER (OPTIONAL)

2) MORE THAN 16 TERMINALS

10 PROCESSOR INTERCONNECT
11 PROCESSOR INTERCONNECT
12 TIME BASE GENERATOR
13 1st MULTIPLEXOR
14 1st MULTIPLEXOR
15 DATA SET CONTROL FOR 1st MULTIPLEXOR
16 2nd MULTIPLEXOR
17 2nd MULTIPLEXOR
20 DATA SET CONTROL FOR 2nd MULTIPLEXOR
21 LINE PRINTER (OPTIONAL)
DIRECTORY

The directory is a table which contains all necessary information about each program or file in the system library. It resides on the drum and may occupy from 1 to 80 drum tracks, depending upon how many discs there are on the system and how many directory tracks per disc are specified by the operator at load time. A core resident table called DIREC contains information on the directory itself.

A directory entry consists of 12 words and has the following format:

WORD 0 user id
1 program or file
2 name
3 start of program pointer for programs/record size for files
4 last reference date (year in bits 9 to 15 day in bits 0 to 8)
5 last change date (hour of year)
6 drum address (0 if not SANCTIFIED)
7 disc address
8 used only by loader
9 length
10 ( - words for program + records for file)
11 BIT 15 = 1 if protected, 0 if unprotected.
BIT 15 = 1 if file, 0 if program
BIT 15 = 1 if semi-compiled, 0 if uncompiled

The directory entries are kept sorted on words 0-3. BIT 15 of words 1 and 2 and 3 are not considered in the sorting. Names of fewer than 6 characters are filled out with spaces (40)
The last reference date is the most recent date on which the program or file was referred to, while the last change date is the most recent date on which it was altered.

The directory contains 2 pseudo entries which are the first and last entries in the table. They have the following form:

<table>
<thead>
<tr>
<th>FIRST ENTRY</th>
<th>LAST ENTRY</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>-1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>-1</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

When the directory occupies more than one track, all the directory tracks appended together form the directory.
II. DIREC

DIREC is a 560 word core resident table which contains information about the directory. It has the following structure:

<table>
<thead>
<tr>
<th>WORD</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>length in words of first directory track</td>
</tr>
<tr>
<td>1-4</td>
<td>same as first 4 words of first directory track</td>
</tr>
<tr>
<td>5</td>
<td>unused</td>
</tr>
<tr>
<td>6</td>
<td>drum address of first directory track</td>
</tr>
<tr>
<td>7-13</td>
<td>same as 0-6 but applied to 2nd directory track</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>553-559</td>
<td>same as 0-6 but applied to 80th directory track</td>
</tr>
</tbody>
</table>

A drum address of 0 implies that there is no such directory track. When word 0 is 0, words 1-4 are meaningless.

The drum address of a directory is always sector 0 of a track. Each directory track may contain as many as 8184 words = 682 directory entries.

When loading the system from paper tape or large tape, the operator has the opportunity to specify the number of directory tracks per disc, in the range of 1-10, which is saved in NDIRT. The total number of directory tracks is this number times the number of discs on the system.
III. ID TABLE

The ID table (IDT) is a drum resident table of from 1 to 3 tracks which contains one 8-word entry for each ID code on the system. The entries are kept sorted according to the ID codes. An entry has the following format:

WORD 0 user id
1-3 password (filled with 0's if fewer than 6 characters)
4 time allowed (in minutes)
5 time used (in minutes)
6 disc allowed (in blocks)
7 disc used (in blocks)

Words 4-7 are 16 bit quantities with values between 0 and 65535.

The 9-word IDEC portion of the EQT has the following format:

WORD 0 first id of first track
1 drum address of first track
2 length in -words of first track
3-5 same as 0-2 but applied to 2nd track
6-8 same as 0-2 but applied to 3rd track

When loading the system from paper tape or mag tape, the operator has the opportunity to specify the number of id tracks, in the range of 1-3, which is saved in NIDT. Each track may contain as many as 8192 words = 1024 id entries.
IV. AVAILABLE DRUM TABLE

The available drum table (ADT) is a drum resident table which contains one two-word entry for each area of the drum which is unallocated. An entry has the following form:

WORD 0 drum address
1 length of area in sectors

Entries are sorted according to word 0. Each entry may refer to as much as one full track, and no two consecutive entries ever refer to two adjacent drum areas (two tracks are not considered to be adjacent).

At the end of the ADT is one additional entry having the form:

0 177777
1 0

The following two memory locations refer to the ADT:

ADLOC = drum address of ADT
ADLEN = length in-words of ADT
V. DISC ADT

The available disc table (Disc ADT) is a drum resident table which contains one three-word entry for each area of the disc which is unallocated. An entry has the following form:

WORD 0 disc
1 address
2 length of area in blocks

There is one Disc ADT track for each disc on the system and only entries for one particular disc appear on a track. The first 4 blocks of each disc are used by the system and are therefore always unavailable. Thus there are no contiguous areas which overlap discs.

The following words in the EQT refer to the Disc ADT:

DADLC BSS 8 drum addresses of Disc ADT tracks
DADLN BSS 8 lengths in -words of Disc ADT tracks

The first word of each BSS refers to logical disc 0, the second to logical disc 1, etc.
VI. LOCKED BLOCKS TABLE

The Locked Blocks Table is a disc-resident table which resides in the 256 words of block 3 of each disc. It contains one two-word entry for each area of the disc that has been MLOCKED. An entry has the following format:

WORD 0 disc address relative to this disc
1 length of area in blocks

The rest of the table is zero filled.

The disc address is stored as if the disc were logical disc 0, so that packs may be used as any logical disc. The Locked Blocks Table is cleared only when it is determined during the loading procedure that the disc does not have a valid TSB label and the operator requests that one be written. This means that the packs "remember" which blocks are unavailable even if different 2000C systems are loaded.
The FUSS table is a 1024 word table which resides on the drum. Its drum address can be obtained by the instruction:

LDA FUSS,1

FUSS is divided into 32 sections of 32 words each. The 32 words in each section are the 2-word disc addresses of the user files currently being accessed by the user corresponding to that table. Addresses of 0 indicate no file. Disc addresses with bit 15 of the first word = 1 indicate that the user has read only access.

The purpose of maintaining this table is to:

1. Prevent simultaneous write access by two users to one file;
2. Prevent moving or removing a file in the routines KILL, LOCK, MLOCK, SANCTIFY and DESECRATE when some user has access to it.

A user's FUSS (i.e. his area of the FUSS table) is set by the FILES routine, which is called from BASIC at the beginning of execution of a program containing a FILES statement. Individual entries in a user's FUSS are changed by the execution of ASSIGN statements. It is cleared by BYE, HELLO, KILLID, and sometimes by KILL.
VIII. COMTABLE

The COMTABLE is a list of all user and system commands containing their ASCII codings and drum locations or core addresses. The structure of the COMTABLE is as follows:

- **COM1**: codes for commands which are executed immediately by the system
- **COM2**: codes for commands which are executed by BASIC
- **COM3**: user commands which are executed by drum resident programs
- **COM4**: system commands — all are executed by drum resident programs
- **COM5**: starting addresses for those commands which are listed under COM1 and COM2
- **COM6**: drum addresses for those commands which are listed under COM3 and COM4 (this section is filled by the loader)

Since each command is recognized only by its first 3 letters, the scanner converts each letter into a number from 0 to 31, and then packs the three codes into one word as three 5-bit bytes. In addition, bit 15 is set for system commands. Codes of -1 in sections 2, 3, and 4 do not correspond to any possible 3-letter code. Their purpose is to generate room in COM6 for disc addresses of routines that are called indirectly, or for tables like FUSS. In the case of CTAPR, the purpose is to generate a status type for printing compiler tape errors without a direct command from the user. Similarly, UCDAB generates a status type for updating the last change date for files after a program is aborted.
LOGGR

LOGGR is a 64-word queue which contains codes for printing LOGON/OFF messages. Entries are placed on the queue by HELLO, BYE, and SLEEP. Each entry consists of 2 words, with the following format:

WORD Ø: user id (BIT 15=0 for ON, 1 for OFF)

1: bits 15-5 = 60 x hrs + mins
   bits 4-0 = terminal number

The representation of a user id is as follows:

BITS 14-10 = letter (A = 1, B = 2, ..., Z = 32)
BITS 9-0 = number (0-999)

The following variables are relevant:

LOGCT = # of unprocessed entries in LOGGR
LOGP1 = points to word 1 of last processed entry
LOGP2 = points to word 1 of last unprocessed entry

Note that LOGCT = 0 <-> LOGP1=LOGP2
X. TELETYPE TABLES

This set of 32 tables, one for each user, contains relevant information about the various terminals. The structure of the tables is as follows:

<table>
<thead>
<tr>
<th>WORD</th>
<th>0</th>
<th>FLAG</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
<td>TNUM</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>DISC</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>PROG</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>ID</td>
</tr>
<tr>
<td>5-7</td>
<td>NAME</td>
<td></td>
</tr>
<tr>
<td>8-9</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>CLOC</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>RSTR</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>STAT</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>LINK</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>PLEV</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>RTIM</td>
<td></td>
</tr>
<tr>
<td>16-20</td>
<td>TEMP</td>
<td></td>
</tr>
</tbody>
</table>

**FLAG:** bits 0-8 contain information as follows:

<table>
<thead>
<tr>
<th>BIT</th>
<th>NAME</th>
<th>MEANING IF = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>TERR</td>
<td>errors while reading program in tape mode</td>
</tr>
<tr>
<td>1</td>
<td>CFLG</td>
<td>program is compiled or semi-compiled</td>
</tr>
<tr>
<td>2</td>
<td>HFLG</td>
<td>$HELLO is running</td>
</tr>
<tr>
<td>3</td>
<td>TAPEF</td>
<td>user in tape mode</td>
</tr>
<tr>
<td>4</td>
<td>UNABT</td>
<td>unable to abort</td>
</tr>
<tr>
<td>5</td>
<td>OUTWT</td>
<td>user suspended for output wait</td>
</tr>
<tr>
<td>6</td>
<td>COM14</td>
<td>communication from I/O processor</td>
</tr>
<tr>
<td>7</td>
<td>ABTRY</td>
<td>abort attempted (while UNABT = 1)</td>
</tr>
<tr>
<td>8</td>
<td>CHNFG</td>
<td>program was called from &lt;CHAIN statement&gt;</td>
</tr>
<tr>
<td>9</td>
<td>ENDST</td>
<td>error on drum transfer</td>
</tr>
<tr>
<td>10</td>
<td>MBUST</td>
<td>error on disc transfer</td>
</tr>
</tbody>
</table>

**TNUM:** teletype number in bits 12-8; used for sending information to I/O processor.
DISC: drum address of user's swap area

PROG: when user is on the drum PROG points to the last core location used by the program. When the user is loaded into core, PROG is placed into PBPTR. When he is written back to drum, PBPTR is copied into PROG. BASIC is required to maintain PBPTR as a bound on the core it is using.

ID: user's id, 0 if none

NAME: a three word entry containing the user's program name. It is set by the routine NAME & GET & CHAIN, and cleared by HELLO. When fewer than 6 characters are in the name, blanks are appended.

CLOC: this is the timeout clock used to determine the length of a user's time slice. See the discussion on scheduling for further information.

RSTR: this is set, when a user is placed on the queue, to his starting address in core. When the user is actually initiated, RSTR is set to 0. Whenever RSTR = 0, the transfer address of the user can be found in location PREG.

STAT: indicates user's status. The user's status is as follows:

-4 port unavailable
-3 enter timeout
-2 system disconnect
-1 user abort request
0 idle
1 system abort
2 input wait
3 output wait
4 syntax processing
>4 command processing

When a command is being processed, STAT indicates the command.

STAT values are assigned in order of entries in the COMTABLE, so that

RUN = 5
LIST = 6
PUNCH = 7, etc.

LINK: the LINK words in the tables are used to form a queue of active users. All users whose status is >4 are in the queue. See discussion on scheduling for further information.
PLEV: this word gives the priority level of the user when he is on the queue. When the user's status is set to 2 or 3, the previous value of STAT is copied into PLEV, and the user removed from the queue. The possible values of PLEV are as follows:

0: highest priority, used for syntax, users returning from I/O suspend, and for disc resident routines once they begin. This includes FILES, CHAIN, and ASSIGN.

1: used for commands RUN, LIST, PUNCH, XPUNCH.

2: used for disc resident routines until they reach the top of the queue.

4: used for long running programs.

RTIM: the length of time in seconds that it took the user to respond to an <ENTER statement>.

TEMP: used (along with RTIM) to save variables when OPEN, CATALOG, GROUP, LIBRARY, STATUS, DIRECTORY, SDIRECTORY and REPORT are swapped out.

Associated with each item in these tables is a symbol which is EQUated to the corresponding number of the item. For example:

?FLAG EQU 0
?TNUM EQU 1
?TEMP EQU 16

These symbols are primarily used for adjusting pointers to the table. For example, if the B register contains a pointer to the LINK entry of some user, the instruction

ADB .+? ID - ? LINK

will point B to his ID entry.

. is a symbol located in base page at the 0 entry of a table of constants from -26 to +49. A word containing the value N, where \(-26 \leq N \leq 49\) can be referenced by .+N.
XI. EQUIMENT TABLE

The equipment table is the area of core which describes the resources available to the system. It resides in locations 100-204, as follows:

<table>
<thead>
<tr>
<th>Locations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100-110</td>
<td>10 table headers (see 111)</td>
</tr>
<tr>
<td>111</td>
<td>number of 10 Tracks</td>
</tr>
<tr>
<td>112</td>
<td>drum address of ADT (see IV)</td>
</tr>
<tr>
<td>113</td>
<td>length of ADT in -words</td>
</tr>
<tr>
<td>114</td>
<td>number of directory tracks per disc (see 11)</td>
</tr>
<tr>
<td>115-124</td>
<td>Disc ADT drum addresses (see V)</td>
</tr>
<tr>
<td>125-134</td>
<td>Disc ADT lengths in -words</td>
</tr>
<tr>
<td>135-140</td>
<td>There is one word in this area for each of the 4 drums. When the word is zero, the particular drum does not exist. Otherwise, bits 7:6 contain the drum prefix and bits 5:0 the high priority select code. The prefix is used by the drum driver as the high order 2 bits of the 8-bit track address.</td>
</tr>
<tr>
<td>141-150</td>
<td>There is one word in this area for each of the 8 discs. When the word is zero, the particular disc does not exist. Otherwise, bits 15:8 contain the high priority select code and 7:0 the unit number</td>
</tr>
<tr>
<td>151-170</td>
<td>This is a table of which drum tracks are physically available to the system. Locations 151-154 correspond to drum 0, 155-160 to drum 1, etc. Track 0 of drum 0 is represented by bit 0 of 151, track 1 of drum 0 is represented by bit 1 of 151, etc. A bit is 0 when the track is available, 1 when unavailable.</td>
</tr>
</tbody>
</table>
The TRAX table is changed only by the following commands:

DRUM - causes all tracks of the specified drum to be made available.

LOCK - all specified tracks are made unavailable.

UNLOCK - all specified tracks are made available.

171-175 SYSID  A ten character system identification. It is set in response to the "SYSTEM IDENTIFICATION?" question on paper tape and mag tape loads. It is used in the headers for STATUS, REPORT, DIRECTORY and SDIRECTORY.

176 MAGSC  High priority select code for mag tape; if non-existent, MAGSC=0. If bit 15=1, the tape unit is a 7970.

177 NPORT  Two's complement of the number of ports on the system. The ports available are numbers 0 thru -NPORT -1.

200 YEAR  Year of the century.

201-202 DATIM  Time of year. The first word is the hour of the year, and the second is the number of 100 ms units in the hour minus 36000.

203 HDATE  Hour of year that the system was last hibernated.

204 SLEPT  0 says that the system has been slept, -1 that it has not. This word is modified only by the sleep and reload procedures and insures that the system may not be reloaded from disc if it has not been slept.
Following the equipment table, in locations 205-216 are another set of words which must correspond with the loader. They are defined as follows:

<table>
<thead>
<tr>
<th>Location</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>205</td>
<td>LDBSA</td>
<td>Core address in the loader of the disc bootstrap.</td>
</tr>
<tr>
<td>206</td>
<td>LSTDA</td>
<td>Core address of the first loader segment in the System Segment Table (SST).</td>
</tr>
<tr>
<td>207</td>
<td>DATLN</td>
<td>Length of the Disc Allocation Table (DAT) in words.</td>
</tr>
<tr>
<td>210</td>
<td>MHAD</td>
<td>Core address of the Moving Head Disc Table (MHTBL)</td>
</tr>
<tr>
<td>211</td>
<td>GMQBP</td>
<td>Core address of the routine to get a buffer for disc or drum error messages. Two such routines exist: one for the loader and one for the system.</td>
</tr>
<tr>
<td>212</td>
<td>DISCA</td>
<td>Core address of disc driver entry point.</td>
</tr>
<tr>
<td>213</td>
<td>DISCB</td>
<td>Core address of disc driver interrupt entry point.</td>
</tr>
<tr>
<td>214</td>
<td>MBUSY</td>
<td>Disc driver busy flag.</td>
</tr>
<tr>
<td>215</td>
<td>MWORD</td>
<td>Word count for disc driver.</td>
</tr>
<tr>
<td>216</td>
<td>DREDP</td>
<td>Core address of disc driver auto restart entry point (used by powerfail/auto restart routine).</td>
</tr>
</tbody>
</table>
XII. SYSTEM SEGMENT TABLE

The System Segment Table (SST) is a 53-word table resident in the loader. It is the first portion of the bootstrap and is pointed to by LDBSA. The first word of the table contains the number of system segments. Each group of 4 words following the first word has the following format:

WORD 1 length of segment in words
2 absolute, beginning core address of the segment
3-4 disc address of the segment

There are 13 segments, ordered as follows:

SEGMENT 1 Interrupt locations (28 to 278)
2 System base page (end of EQT to 17778)
3 System linkage area (20028 to 20158)
4 System segment 1 (end of Direc to 41778)
5 " " 2 (420008 to 51778)
6 " " 3 (520008 to 61778)
7 " " 4 (620008 to 71778)
8 " " 5 (720008 to 776778)
9 Equipment table (1008 to 2168)
10 Direc table (300008 to 310578)
11 Loader segment 2 (160008 to 25778)
12 Loader Segment 1 (20008 to 146778)
13 Disc driver (260008 to 27778)

Note that this includes all core resident portions of the loader and system except for locations 147008 to 15778. The first 1008 of these words comprise the disc bootstrap and are resident on blocks 1 and 2 of each disc. Locations 157008 to 15778 may only be used for temporaries.
XIII. DISC ALLOCATION TABLE

The Disc Allocation Table (DAT) is a 276-word table resident in the loader. It is the first portion of Loader Segment 2 and its disc address is pointed to by MEM[LSTDA] + 2 when the SST is in core. The DAT designates the areas allocated on the disc for storage (during a SLEEP or HIBERNATE) of system library programs and system tables normally resident on the drum.

There are 4 sections of the DAT. The first (DATSL) is a 3-word entry consisting of the length in blocks of the system library and the 2-word disc address of the first system library program.

The other 3 sections (DATID, DATDA, and DATDI) contain one 3-word entry for each reserved area on the disc for the Id table, Disc ADT, and Directory tracks respectively. The format of these entries is the length in -words in the first word and the disc address in the second 2 words. For these sections 32 blocks are always reserved for each track, since the tables may grow to this size while the system is running.
XIV. MOVING HEAD DISC TABLE

The Moving Head Disc Table (MHTBL) is a 48 word table resident in the disc driver section of the loader and system. It contains hardware information about the disc on the system as follows:

WORD 0-1 Two-word logical address of the first 128-word hardware sector on logical disc 0.

2 Points to select code/unit number in DKTBL for logical disc 0

3 number of sectors/cylinder

4 number of sectors/track

5 Current cylinder position of heads for logical disc 0 (not used for 2883 discs)

6-11 Same as 0-5 applied to logical disc 1

::

42-47 Same as 0-5 applied to logical disc 7

Note that the address in the first two words of each section of the table is a sector address and must be divided by 2 to obtain the block address.

The actual numbers for the 3 kinds of used on the 2000C are as follows:

<table>
<thead>
<tr>
<th></th>
<th>2883</th>
<th>2870</th>
<th>7900</th>
</tr>
</thead>
<tbody>
<tr>
<td>WORDS 1-2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>93380</td>
<td>9744</td>
<td>19488</td>
</tr>
<tr>
<td></td>
<td>18676</td>
<td>19488</td>
<td>38976</td>
</tr>
<tr>
<td></td>
<td>280140</td>
<td>29232</td>
<td>58464</td>
</tr>
<tr>
<td></td>
<td>373520</td>
<td>38976</td>
<td>77952</td>
</tr>
<tr>
<td></td>
<td>466900</td>
<td>48720</td>
<td>97440</td>
</tr>
<tr>
<td></td>
<td>560280</td>
<td>58464</td>
<td>116928</td>
</tr>
<tr>
<td></td>
<td>653660</td>
<td>68208</td>
<td>136416</td>
</tr>
<tr>
<td>3</td>
<td>460</td>
<td>48</td>
<td>96</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>12</td>
<td>24</td>
</tr>
</tbody>
</table>

20
## CORE MAP

<table>
<thead>
<tr>
<th>OCTAL LOCATION</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>INTERRUPT LINKAGE AND UNINITIALIZED SYSTEM VARIABLES</td>
</tr>
<tr>
<td>100</td>
<td>EQUIPMENT TABLE</td>
</tr>
<tr>
<td>217</td>
<td>CONSTANTS AND SYSTEM VARIABLES</td>
</tr>
<tr>
<td>USER 1224</td>
<td>REGISTERS SAVED BY CLOCK</td>
</tr>
<tr>
<td>LIBUS 1230</td>
<td>USER SWAP AREA AND SYSTEM LIBRARY WORK AREA (10240 WORDS)</td>
</tr>
<tr>
<td>26000</td>
<td>DISC DRIVER</td>
</tr>
<tr>
<td>30000</td>
<td>DIREC TABLE</td>
</tr>
<tr>
<td>31060</td>
<td>DLOOK ROUTINES</td>
</tr>
<tr>
<td>31701</td>
<td>BASIC</td>
</tr>
<tr>
<td>60000</td>
<td>I/O DRIVERS</td>
</tr>
<tr>
<td>61124</td>
<td>TELETYPETABLES</td>
</tr>
<tr>
<td>62264</td>
<td>EXECUTIVE</td>
</tr>
<tr>
<td>71624</td>
<td>COMMAND TABLE</td>
</tr>
<tr>
<td>72140</td>
<td>SYSTEM LIBRARY SUBROUTINES</td>
</tr>
<tr>
<td>LIBRA 75000</td>
<td>SYSTEM LIBRARY PROGRAMS SWAP AREA (512 WORDS)</td>
</tr>
<tr>
<td>77000</td>
<td>CORE DUMP</td>
</tr>
<tr>
<td>77700</td>
<td>PROTECTED LOADER</td>
</tr>
</tbody>
</table>
DRUM ORGANIZATION

The drum space available to the system consists of from 64 to 256 tracks, depending upon how many drums exist. Each track contains 128 sectors of 64 words each, for a total of 8192 words per track. The loader assigns tracks as follows:

- System library routines (3 tracks)
- IDT (1-3 tracks)
- User swap tracks (1 1/4 - 40 tracks)
- Directory (1-80 tracks)
- Disc ADT (1-8 tracks)
- ADT (1 track)

All remaining tracks are available for storage of sanctified user programs and files. The ADT contains an entry for each available area.

The drum addresses of the individual system library routines are stored into the COMTABLE during loading. Although they are not all the same length, they are limited to 512 words, and so the system reads in exactly 512 words whenever it wants to load such a routine. The loader never assigns a library routine within 7 sectors of the end of a drum track, so that no errors can take place in doing this.

Each directory, IDT, ADT and Disc ADT track is stored beginning at sector 0.

Since the user area is 10240 words long, it cannot fit on a single track. The loader must therefore find adjacent areas on the drum which total 1 1/4 tracks for each user swap area. This will not cause any problems because all drums used on the 2000C have automatic track switching.
During running, each user swap area contains a copy of the area from core location USER through the core location specified by its ?PROG entry. This includes all variable data which is relevant to that user's program, and his program itself. The location of various sections in his program is discussed elsewhere.

Programs and files which are designated as SANCTIFIED by the operator reside on the drum and thus have better access time. They must be less than 8192 words long. The area on the disc where they were originally resident is reserved so they may be copied back at sleep time.
DISC ORGANIZATION

The disc space available to the system is determined by the number and type of discs which exist on the system. The discs are divided into 256 word blocks. There are 46690 such blocks on a 2883 disc, 9744 on a 7900 disc, and 4872 on a 2870 disc.

The first 4 blocks of each disc are reserved for use by the system. Block 0 is a label, which looks like this:

WORD 0 "LB"
1 "TS"
2 logical disc number
3-7 system identification
8-30 ∅
31 checksum of words 0-30

Blocks 1 and 2 contain the final disc bootstrap found in the loader, and block 3 contains the locked blocks table for this disc, which is discussed elsewhere.

Disc space for system usage is assigned as follows:

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resident system</td>
<td>130 blocks</td>
</tr>
<tr>
<td>System library</td>
<td>126 blocks</td>
</tr>
<tr>
<td>IDT</td>
<td>32 blocks/track</td>
</tr>
<tr>
<td>Disc ADT</td>
<td>32 blocks/track</td>
</tr>
<tr>
<td>Directory</td>
<td>32 blocks/track</td>
</tr>
</tbody>
</table>

All remaining blocks are available for storage of user programs and files. Programs and files are each required to be stored as contiguous blocks of disc. Since the disc is allocated by blocks, each program may cause part of its last block to be wasted. When a program is stored (by the SAVE routine), it is first decompiled and is stored in that form. Only the encoded text is stored, so that a program may require as little as 3 words of disc space. When a program is stored (by the CSAVE routine) it is saved in a semi-compiled form, i.e. the form it is in after the symbol table is built. Both the encoded text and the symbol table are stored, plus 7 words of necessary information.
Files always occupy an integral number of records (1-32767), each file occupying a contiguous area on the disc. BASIC does not treat the individual records in the same logical sequence as the physical sequence, but rather interleaves the records, as follows:

**even # of records**

Physical sequence: 1 2 3 4 ... 2n-2 2n-1 2n
Logical sequence: 1 n+1 2 n+2 ... 2n-1 n 2n

**odd # of records**

Physical sequence: 1 2 3 4 ... 2n-2 2n-1
Logical sequence: 1 n+1 2 n+2 ... 2n-1 n

This format tends to decrease disc seek time when records are accessed in a logically ascending order.
DISC AND DRUM ERROR ROUTINES

Disc and drum errors do not cause immediate halts in their respective drivers. Instead, the drivers indicate failures to their calling routines, which take appropriate action. In many cases (particularly most disc errors), the action is merely to inform the affected party and continue normal system operation. Such is the case for all user access to programs and files on the disc or drum.

In certain cases, however, system operation is more significantly affected. The following routines are called after these disc and drum failures:

JETPT

This routine is called when a drum transfer to or from a user's swap area has failed. It proceeds as follows:

1. Remove the user from the queue.
2. Set the port's status to unavailable and clear its ID and flags words.
3. Clear this user's area of the FUSS table.
4. Call TCRIR to inform the user that his port is being zapped.
5. Insert an informative message into the system console message queue and return.

SALVG

This routine is called when a system track with vital information (such as a directory or IDT track) cannot be written back to its assigned drum address, but when recovery might be possible if the information can be saved. It is entered with A containing the address of the word in core which contains the drum address of the track in question. B contains the negative length of the table, which must be in core starting at LIBUS. The operation is as follows:

1. Read the drum ADT, in several portions if necessary, into the upper 2K of user area. Search each piece for an entry large enough to accommodate the table in core. If none is found, go to step 4.
2. Attempt to write the table to the newly found area on the drum. If the write is unsuccessful, go to step 4.

3. If the table is successfully written to the drum, update the drum address in core associated with it, call CLNOT to finish printing any messages in the system message queue as well as a message of success, and halt.

4. Call CLNOT to clean out the message queue and to print a message of SALVG's failure, and then halt.

**SICK**

This routine is called when the system cannot continue operating (because, for example, it cannot read a library routine or system table), but may be able to be recovered. The routine merely calls CLNOT to finish printing any messages in the queue and a message of hope, and then halts.

**DEAD**

This routine is called when the system cannot continue operating and has altered its tables in such a way that they contain conflicting information, and recovery is, therefore, impossible. DEAD calls CLNOT to dump the message queue and output a "no recovery" message, and then halts.

**MDEAD**

This routine is called in situations like those which call DEAD, but which specifically involve possible destruction of a disc's locked blocks table. The procedure is the same as that in DEAD, except that the final message refers to the locked blocks table.

**CLNOT**

This routine is called when a hardware error has caused a system shutdown to be initiated and it is desired to inform the operator and users of what is happening. The procedure is:

1. Call TCRIR to inform all users that the system is going down.
2. If the system teletype driver was outputting, complete the line it was printing.
3. If there are messages in the queue, output them on the system console.

4. If the routine was entered with B nonzero, print the message whose length and text are pointed to by it.

5. Output three X-OFF CR LF's, and return.

TCRIR

This routine is called to inform one or all users of a hardware failure which is fatal to him/them. The procedure is:

1. Set up port counts and message pointer for the appropriate message.

2. Output the message, one character at a time, to each port to receive it.

3. Output three linefeeds and return.

SYCON

This routine is the non-interrupt output-only teletype driver used by CLNOT to print on the system console. Upon entry, A holds the number of characters to be output. Bit 15 of A = 0 for X-OFF CR LF to be output after the line. B points to the first word of the buffer to be output.
SCHEDULING

The basic philosophy of the TSB scheduling algorithm is to provide short response times for short, interactive jobs at the possible cost of delays in longer running jobs. The implementation of this involves a queue of jobs to run which is ordered according to a priority scheme. The queue is a linked list of from 1 to 34 entries, each entry pointing to the next entry, and the last entry pointing back to the first. The 34 possible entries in the queue are the 32 user LINK entries, a LINK word in a truncated TELETYPE table reserved for the system console, and a queue head. The queue head consists of the locations MLINK (0:2), and is always in the queue. The queue head has a priority of 777778, which is stored in location MLINK+2, and so it is always the last entry in the queue. As an example of how this works, assume that users 1, 3 and 6 are on the queue in that order and so is the system console, in a position between users 3 and 6. Then the queue will have the following appearance:

```
| TTY0+?LINK | 0 |
| TTY3+?LINK | 2 |
| T35LK      | 2 |
| T35PR      | 4 |
| TTY6+?LINK | 4 |
| MLINK      | 77777 |
```
Since the MLINK entry is always the last entry on the queue, MLINK+1 is a pointer to the first entry, which in this case is TTY01. In the case of an empty queue, MLINK+1 will point to itself, i.e., CONTENTS(MLINK+1) = CONTENTS(MLINK). Each entry on the queue has a priority no greater in numerical value than that of the one it points to. When an entry is added to the queue, this ordering is always preserved by placing the new entry just ahead of the first entry with a larger priority number. Note that when the first entry in the queue has priority 0, it will remain at the head of the queue until it is removed from the queue entirely.

The following rules are used to assign (and reassign) priorities:

1. Upon first entering the queue, jobs are assigned priorities as follows:
   - SYNTAX lines and jobs returning from I/O suspend: 0
   - BASIC commands (RUN, LIST, PUNCH, XPUNCH): 1
   - Commands for drum-resident routines (GET, BYE, etc): 2

2. Priorities of jobs are reassigned in the following way:
   Jobs of priority 2, when they reach the top of the queue, are reassigned priority 0.

   RUN jobs, when they exceed their time slice, are reassigned priority 4, and repositioned in the queue according to that priority. Each RUN job is assigned a time slice of two seconds, and if it exhausts that it is assigned another. When executing a <CHAIN statement>, a <FILES statement>, or an <ASSIGN statement>, a RUN job is reassigned a priority of Ø.

   The OPEN command is reassigned a priority of 4 when it is suspended after writing file marks in 400 blocks.
After an abort during program execution a user is re-assigned a priority of 0 to run the routine which updates the last change date for files.

LIB points to the location in the COMTABLE of the drum address of the library routine in core. LIB = 0 when none is present.

The following conditions must exist for the scheduler to permit execution:

A) for Syntax and BASIC commands:
   MAIN set to point to correct user table

B) for drum resident commands:
   MAIN = 0
   LIB set to correct drum resident routine

The scheduler routine SWAPR is responsible for creating these conditions, and makes its decisions according to the values of MAIN, LIB, and the entry on top of the queue.
COMMUNICATION BETWEEN SYSTEM MODULES

There are seven system modules that communicate with each other in various ways: the drum driver, the disc driver, I/O Processor driver, system console driver, scheduler, BASIC, and system library routines (HELLO, BYE, KILLID, etc.).

1. Drum Driver.

Any section of the system may call the drum driver to perform a drum transfer. Three parameters are passed:

A = drum address
   (bits (15:14) = drum number
   bits (13:8 ) = track number
   bit 7    = 0
   bits ( 6:0 ) = sector number)

B = core address
   (bits (14:0 ) = core address
   bit 15    = 1 for drum input
             0 for drum output)

WORD = -# of words to be transferred (may be 0, in which case no actual transfer is performed).

Called by JSB DRUM,1

It is the responsibility of the caller to insure that the drum is not busy when the call takes place. This is no hardship since while BASIC or a system library routine is running, no other module even initiates drum transfers. As a result, the drum will appear to be busy only if the module itself has initiated the transfer.

Upon initiation of a drum transfer, the variable ENDRM is set to -1, and it is cleared upon completion. A complete transfer can be performed by:
JSB DRUM,1
LDA ENDRM
SSA
JMP *-2
SZA
JMP <error location>
<process successful transfer>

The system never suspends a program for a drum transfer because the high speed of the drum does not cause any great overhead.

The value of WORD is not modified by the driver.
II. Disc Driver

Any section of the system may call the disc driver to perform a (moving-head) disc transfer. Three parameters are passed to the driver:

\[
\begin{align*}
A &= \text{pointer to disc address} \\
B &= \text{core address} \\
\end{align*}
\]

\(A\) and \(B\) are described as follows:

- \(A\): Pointer to the core address of a two-word logical disc block number at which the transfer is to begin. (bits 14-0 = core address at which transfer is to begin; bit 15 = 1 for read from disc to core; bit 15 = 0 for write from core to disc)

- \(B\): Core address

The variable \(\texttt{MWORD} = \) the negative of the number of words to be transferred. If \(\texttt{MWORD} \geq 0\), the driver will cause no transfer, but will position the appropriate disc unit at the specified block.

The disc driver is called by JSB DISCA,1.

The driver determines the logical disc on which the specified block lies, and, if that logical disc is present on the system, processes the requested transfer. While a request is being processed and transfer taking place, the driver busy flag, \(\texttt{MBUSY}\), is set to -1. If the driver is called while \(\texttt{MBUSY}\) is so set, it will return without doing anything. If the disc block number passed to the driver does not lie on one of the discs present on the system, the driver will increment the return address by one and return without doing anything. If the driver accepts the request, it will increment the return address by two and return after processing of the request has been initiated.
A moving head disc transfer involves two steps: positioning the heads to the correct area of the disc and performing the actual data transfer. The disc driver returns to its caller while each of these is going on. Command channel interrupts return control to the driver when the operations are complete; the driver checks for successful completion of the operations before proceeding.

The driver for the 2870 (IOMEC) disc keeps track of the cylinder position of the heads on each of the discs on the system. If a requested transfer is to or from the current cylinder of a disc, the driver does not issue a seek command and suspend pending its completion before starting the read or write. It merely issues an "address record" command to set the disc controller's record address register for the transfer. The 2883-2884 (ISS) disc driver always issues a seek command, since a seek to the current cylinder consumes virtually no time.

A single data transfer on a disc cannot automatically continue from one cylinder to the next. The 2870 disc has the further restriction that a transfer cannot cross the 'mid-cylinder' boundary (between track 1 and track 2). When a data transfer is requested which crosses one or more of these boundaries, the disc driver breaks up the transfer to conform with the restrictions.

When the driver completes handling a request and returns to the caller, MBUSY is set to indicate the outcome of the transfer as follows:

0: the requested transfer has been successfully completed.

1: the transfer has failed; the seek (position) operation could not be completed.
2: the transfer has failed; the data transfer was unsuccessful.

3: the transfer has failed; part of the data lies on, or would be written to, a disc which is not present on the system.

A complete disc transfer can be performed by the following sequence:

```
JSB DISCA,1
<return for driver busy>
<return for disc not present>
LDA MBUSY
SSA
JMP *-2
SZA
<process disc error>
<process successful transfer>
```

The disc driver does not modify the contents of MWORD and the A and B registers. The system never suspends a program for a disc transfer.
III. I/O PROCESSOR DRIVERS

There are two drivers used for communication between the main processor and the I/O processor, one for each direction of communication. Each communication consists of one 16 bit word which looks like this:

```
15 13 12  8 7  4 0
OPCODE   TTY #  DATA OR OPCODE
```

The TTY # is the user's port number as found in the ?TNUM word of his teletype table.

The opcode uses bits 15-13, unless they are all 1's, in which case it also uses bits 4-0.

Opcodes which have values of bits 15-13<4<8 use bits 7-0 for data, e.g. al ASCII character.

The main processor sends communications on I/O channel 11 and receives them on I/O channel 10. An exception is a communication sent by the main processor which requires a response, which will be received on 11. Communications are initiated by JSB SL4SC, I with the communication in the A register.

The following is a list of communications sent by the main processor:

<table>
<thead>
<tr>
<th>NAME</th>
<th>OPCODE (OCTAL)</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCR</td>
<td>000000</td>
<td>Output a character</td>
</tr>
<tr>
<td>STE</td>
<td>020000</td>
<td>Start timing &lt;ENTER statement&gt;</td>
</tr>
<tr>
<td>GTC</td>
<td>040000</td>
<td>Get a character (response required)</td>
</tr>
<tr>
<td>PHO</td>
<td>060000</td>
<td>Allowed phones time</td>
</tr>
<tr>
<td>SPE</td>
<td>100000</td>
<td>Baud rate information</td>
</tr>
<tr>
<td>SBP</td>
<td>120000</td>
<td>Save teletype buffer pointer</td>
</tr>
<tr>
<td>RBP</td>
<td>140000</td>
<td>Restore teletype buffer pointer</td>
</tr>
<tr>
<td>INI</td>
<td>160000</td>
<td>Initialize system</td>
</tr>
<tr>
<td>UIR</td>
<td>160001</td>
<td>User is running</td>
</tr>
<tr>
<td>UNR</td>
<td>160002</td>
<td>User no longer running</td>
</tr>
<tr>
<td>IWT</td>
<td>160003</td>
<td>User in input wait</td>
</tr>
<tr>
<td>HUU</td>
<td>160004</td>
<td>Disconnect user</td>
</tr>
<tr>
<td>ULO</td>
<td>160005</td>
<td>User logged on successfully</td>
</tr>
<tr>
<td>ECO</td>
<td>160006</td>
<td>Echo-on</td>
</tr>
<tr>
<td>ECF</td>
<td>160007</td>
<td>Echo-off</td>
</tr>
<tr>
<td>NAME</td>
<td>OPCODE (OCTAL)</td>
<td>FUNCTION</td>
</tr>
<tr>
<td>--------</td>
<td>----------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>TPO</td>
<td>160010</td>
<td>User in tape mode</td>
</tr>
<tr>
<td>ILI</td>
<td>160011</td>
<td>Illegal input? (requires response)</td>
</tr>
<tr>
<td>NUC</td>
<td>160012</td>
<td>New user logged on</td>
</tr>
<tr>
<td>KAO</td>
<td>160013</td>
<td>Kill-all output</td>
</tr>
<tr>
<td>ALI</td>
<td>160014</td>
<td>Allow input</td>
</tr>
<tr>
<td>OWT</td>
<td>160015</td>
<td>User in output wait</td>
</tr>
<tr>
<td>IBF</td>
<td>160016</td>
<td>Is buffer full (requires response)</td>
</tr>
<tr>
<td>PSC</td>
<td>160017</td>
<td>Line printer select code</td>
</tr>
<tr>
<td>LPR</td>
<td>160020</td>
<td>Line printer request (requires response)</td>
</tr>
<tr>
<td>LPD</td>
<td>160021</td>
<td>Line printer disconnect</td>
</tr>
<tr>
<td>LPS</td>
<td>160022</td>
<td>Line printer status (requires response)</td>
</tr>
<tr>
<td>BKS</td>
<td>160023</td>
<td>Backspace in teletype buffer</td>
</tr>
<tr>
<td>CHS</td>
<td>160024</td>
<td>Character size information (requires response)</td>
</tr>
<tr>
<td>STP</td>
<td>160025</td>
<td>Subtype information</td>
</tr>
<tr>
<td>WSP</td>
<td>160026</td>
<td>What baud rate (requires response)</td>
</tr>
<tr>
<td>WCS</td>
<td>160027</td>
<td>What character size (requires response)</td>
</tr>
<tr>
<td>WTP</td>
<td>160030</td>
<td>What terminal type (requires response)</td>
</tr>
<tr>
<td>TKO</td>
<td>160031</td>
<td>Telekludge line printer output</td>
</tr>
</tbody>
</table>

Communications initiated by the I/O processor are detailed elsewhere. It should be noted that the main processor ignores communications if they are not consistent, e.g. it will only accept a line of input when the user's status is idle or input wait. The receive driver communicates with the scheduler by setting the COM14 bit in the ?FLAG word of the user's teletype table and setting the appropriate status.

The I/O processor program is responsible for all multiplexor I/O. Output to the multiplexor is performed on a character by character basis via the routine OUTHC. The calling sequence is as follows:

A = character to be output in bits (6:0), bits (15:7) ignored.
B = address of WORD 0 of users teletype table
   (JSB OUTHC, I).

The OUTHC routine places characters into the user's buffer until it is filled (250 characters), at which point the user is suspended by OUTHC. This is no problem for BASIC, but due to re-entrancy problems must not be allowed by other modules. The buffer is always empty when a library routine is initiated, so they normally do not have to worry about it. Routines which may fill the buffer, like CATALOG and DIRECTORY, get around the problem by suspending themselves at an appropriate time.
The I/O processor program recognizes aborts and sends them to the main processor. If the user is running a library program (except CATALOG, LIBRARY, GROUP, DIRECTORY, SDIRECTORY, REPORT and STATUS), the abort is ignored, since the routine may be in the process of updating system tables. At other times when aborting could cause trouble, the UNABT bit in the ?FLAG word of the TTY table is set. When the abort is seen, the ABTRY bit is set. Routines which set UNABT have the responsibility of calling ABCHK when aborts will no longer cause harm. ABCHK aborts the user if ABTRY was set.

Input from a user teletype is buffered by lines. When the I/O processor sees a carriage return, it informs the main processor. BASIC, or the command processor, or the library routine, etc. processes the input on a character by character basis.
IV. SYSTEM CONSOLE DRIVER

The system console driver maintains three flags, T35F1, T35F2, and T35F3, which determine its status. The meaning of these flags are as follows:

T35F1: = -1 during output, 0 otherwise
T35F2: Normally 0, it is set to -1 by the driver at the conclusion of input, and cleared to 0 externally.
T35F3: Normally 0, it is set to -1 by the driver at the conclusion of input, and cleared to 0 by the driver after output has been initiated.

The combined values of these flags are more significant:

<table>
<thead>
<tr>
<th>F1</th>
<th>F2</th>
<th>F3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

When F2 = -1, the driver will not accept any input. This guarantees system library programs that they will not be interfered with. These routines are responsible for clearing F2 when they call the driver for the last time. F2 and the console status (T35ST) are also cleared if a key is struck on the console during output. This will effectively terminate such things as DIREctories, REPORTs and STATuses.

When F3 = -1, log-on and log-off reports as well as the message queue are held off. This guarantees that these messages will not be interfered with by system library program output.
The calling sequence is:

A: bit 15 = 0 if CRLF is to be appended, bits (14:0) = # of chars.
B: bit 15 = 1 if punching is to take place in addition to printing,
   bits (14:0) = core address of output buffer.
   JSB TTY35,1

The driver uses the 36 word buffer T35BF as an input buffer. Most of
the library routines use it for output, and occasionally for temporary
storage between lines of output.

V. INPUT AND TERMINATION REQUESTS

   BASIC may obtain input from a user console by performing the
   instruction
   JSB SCHIN,1

   Either BASIC or a system library routine terminates by:
   JSB SCHEN,1

   It is possible for BASIC to call a system library routine directly
   by executing:
   JSB SCHLB,1
   DEF <location in COMTABLE of drum address of program>

   This is done with the FILES, CHAIN and ASSIGN routines. It is
   necessary that the library routine cooperate with BASIC, i.e., not any
   program can be so called.
CONSOLE DRIVER

1. If T is 18,
   - TCNT = TCNT - 1
   - TDR1C = ?

2. If TCNT = 0 or 3,
   - Yes: TCNT = ?. TBITS = 120000
   - No: TCNT = TCNT + 1

3. If TDR1C = ?
   - Yes: TT8
   - No: TBITS = 120000

4. Output to TTY Select Code

5. Get Next Character

6. TT8
power fail interrupt

power up or power down

SAVE REGISTERS

interrupt out of p.f. routine?

Clear power Flag

Copy saved registers to second save area

Save flags of the I/O channels.

HALT
POW1

RESTORE TBS AND TRY STATUS.

INITIALIZE LOOP

WAS THE FLAG SET ON THIS DEVICE?

YES

POW5

NO

CLEAR FLAG ON THIS DEVICE

ADVANCE TO NEXT DEVICE

CHECKED ALL SEC CODES YET?

NO

YES

POW14
POWER FAIL

1. power

2. Are we servicing the interrupt for this device? [NO/GO]
   - NO: power
   - GO: Set control on this device and allow an interrupt to occur, this will clear the IRQ 0.

3. Clear control on this device

4. Adjust the poluters for the next Check.

5. power
2116 POWER FAIL

In this case the drum device will return to its caller.

RE ENTER DRUM DRIVER DIRECTLY

REDO DISK AUTO RESTART ROUTINE

THE DISK DRIVER EXITS TO ITS CALLER IF THE POWER FAIL TIME PLACE INSIDE THE DISK TRANSFER.

REDO DISK AUTO RESTART ROUTINE

RESTORE REGISTERS AND INTERRUPT SYSTEM
DREDY

SAVE REGISTERS; DISABLE INTERRUPTS; SET FOR DISC 0

SELECT CODE = ?

WAIT FOR DISC READY

MOVE TO NEXT DISC

ALL 8 DISCS DONE?

WAS DISC DRIVER BUSY?

SET UP PARAMETERS FOR DRIVER

RE-ENTER DRIVE DIRECTLY TO RESTART TRANSFER

RESTORE REGISTERS

RETURN TO AUTO-RESTART Routine

SET FOR RESTART BACK HERE; SET UP PARAMETERS FOR DRIVER

RE-ENTER DRIVE TO RESTART TRANSFER
COMMUNICATION DRIVERS — SEND

S14SH

Interrupt system off

Save state of clock and blank it.

Interrupt system on

Is processor ready for another I/O?

Yes

Send down the word.

No

Interrupt system off

Resume state of clock (blanks or unlocked)

Interrupt system may resume, send etc.
COMMUNICATIONS DRIVERS -- RECEIVE

RI4 CM

Save registers A, B, E, O, P

Get the communications word

Jump to 1243

Decode of the opcode

ANET

Restore registers A, B, E, O

Acknowledge receipt to 316 processor

Restore P
COMMUNICATIONS DRIVERS -- RECEIVE

```
HVL

FDDTY

Is status file or input wait?

Set this user come bit.

Save lower 9 bits in RT2M in case response for ENTER.

RINT
```

User has typed a carriage return.
CLOCK INTERRUPT

UPDATE THE TIME OF DAY, HOUR OF YEAR

EVEN OR ODD INTERRUPT?

IS THE CLOCK DISABLED?

NO

SAVE REGISTERS AND BEGIN THE CLOCK

CLF CLOCK

YES

EXIT

CLF CLOCK

IS THIS A TIMED PROGRAM?

NO

SCH1

YES

BEGIN PROCEDURE

END

priority and remove from Q

SCH2
SCHEDULER

SET POINTER BACK TO FIRST TTY TABLE

CONSOLE DRIVER BUSY?

ANY ENTRIES IN MESSAGE Q?

UPDATE COUNT AND MESSAGE BUFFER POINTER

TTY 35
OUTPUT A MESSAGE

ANY LOG MESSAGES?

UPDATE COUNT AND BUILD LOG MESSAGE

TTY 35
OUTPUT A LOG MESSAGE

SCHEDULER
If input has been typed in, output has almost completed.

Restore the actual status (saved in PLEV)

Set PLEV = \( \emptyset \)

Adjust pointer for \( \emptyset \) list

Set up for running core resident AUCOC routine

Remove from queue

Remove output bit in flags

Kill output

Need to update last change date?

Set status to zero (idle)

Print "Stop"

HTEST

SCH1
This routine prepares the program on the top of the queue to run.

**S W A P K**

- Is the disk or disc busy? (Y: SCH1)

  - Was swap doing a transfer? (Y: SCH1)

  - Was it successful? (Y: SCH1)

  - Have we saved the flags? (Y: SCH1)

  - Save disk and drum busy flags in try table (N: SCH1)

  - Is the queue empty? (Y: SCH1)

- Is the console output line? (N: SCH1)

  - Can the routine be defered? (Y: SCH1)

  - Type III? (Y: SCH1)

- Any program in core? (Y: SCH1)

  - Write out program in core? (N: SCH1)

  - Write out program in core? (Y: SCH1)

  - Return (Y: SCH1)

  - SWAPZ (Y: SCH1)

  - Remove routine from queue (Y: SCH1)

- SWAP (Y: SCH1)
The FILES routine is used by BASIC to process FILES statements in a user's program. The function of the FILES routine is to translate the file names in the user's program into a table for use during execution. This table contains a 15-word entry for each file. Its format is:

<table>
<thead>
<tr>
<th>Word</th>
<th>Description</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>file length in records (Ø for none)</td>
<td>bit 15=1 if read only</td>
</tr>
<tr>
<td>2</td>
<td>logical record size in words</td>
<td>bit 15=1 if &quot;dirty&quot; record</td>
</tr>
<tr>
<td>3</td>
<td>disc or drum address of file's last logical record</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>disc or drum address of record currently in core (word 5 = 1000000 if none)</td>
<td>drum addresses in words 4, 6, and 8 for sanctified files; for these, words 3, 5, and 7 = 1777778</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>disc or drum address of file's first record</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>pointer to first word beyond core buffer</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>pointer to current word in buffer</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>EOF/EOR exit address (Ø for none)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>file</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>name</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>protect mask</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
During operation of the FILES routine, a temporary buffer is used as a table to store intermediate data. Eight words of the buffer are used for each file. The operation is as follows:

1. Translate characters in FILES statements into the buffer table. FILES statements are pointed to by a four word table in the user swap area which is pointed to by DFILT. 

   FILCT = -5 + # of FILES statements. There may be up to 4 such statements. Filenames are extended to six characters, if necessary, and those which are specified to be public files are marked by setting Bit 15 of their first word to 1. Those which are specified to be group library files are marked by setting bit 7 of their first word to 1. A "*" alone as a file name is a place holder. The buffer table for the entry is zeroed. Possible errors found in this step are:

   a. File name of 0 or > 6 characters
   b. More than 16 files requested

2. Perform directory search for each specified file. DIRWD is set to the drum address of the directory track in core so that DLOOK doesn't have to read and write the directory for each file. Save the file's drum address (Ø if not sanctified), disc address, file length, and logical record size in its portion of the buffer table. The record size for "*" entries was set to 256 in the previous step. The read-only bit is set if the file is a library file and the user is not the owner. An error occurs if the file is nonexistent or protected. Update the last reference date word in the directory entry for this file.

3. Test to make sure that there is sufficient room in the user area for the file table.

4. Scan the FUSS table to see if any other user has write capability on the files requested. Mark any such files as read-only. This test is skipped if the user's ID has a letter prefix 'A'. Copy the disc addresses of the requested files into the user's portion of FUSS. Indicate read-only files by setting bit 15 of the upper disc address word.
5. Build the table specified above. FILTB is a pointer to the beginning of the table. Upon exit, VALTB and PBPTR both point to the first word following the table.
ASSIGN

The ASSIGN routine is used by BASIC to process an ASSIGN statement in a user's program. The function of the routine is to replace the information currently in the file table referenced by a specified file number with information about a new (usually different) file being assigned to that number. The operation is as follows:

1. If the previous file was written on, update the last changed date word in its directory entry.

2. Search the directory for the new file. If it is not found, is a program, or is protected, exit to non-existent file return. If it has records larger than those of the previous file, exit to that return location. Otherwise save the file's drum address, disc address, length, and record size. Set bit 15 of the length word (read-only) if this is a user accessing a system or group library file not his own. Update the last reference date word in the directory entry for the file.

3. Scan the FUSS table to see if any other user has write capability on the file. If so, set bit 15 of the file's length word unless this is an "A" user. Move the disc address of the file into the appropriate two words of the user's portion of the FUSS table, setting bit 15 of the upper word if the file is read only for this user.

4. Construct the file table entry specified in the description of the FILES routine. Exit to one of three locations, depending on whether the file is: 1) available for reading and writing; 2) read-only (except users Axxx) because of another terminal's read-write access; or 3) read-only because it is a system or group library file.
The CHAIN routine is used by BASIC to process a CHAIN statement in a user's program. The function of the CHAIN routine is to find the program named in the CHAIN statement, retrieve it from the disc, and begin execution. It operates as follows:

1. Dump file buffers.

2. Update the last changed date entry in the directory for each file which was written on.

3. Translate name of program from CHAIN statement. Invalid names exit to error. If preceded by a "$", set up A$ search; if preceded by "@", set up group library search; otherwise set for searching on user's ID. Save the line number if any is specified.

4. Perform directory search. Exit to error if not found.

5. Check to make sure that the entry is a program, that it is not ill-stored, and that it will fit. If any of these are not true, exit to the appropriate error.

6. For programs on the disc (not sanctified), initiate a seek by calling the disc driver to perform a zero length transfer.

7. Update date entry in directory and write directory track back to drum.

8. Read in the basic portion of the previous program, including the common area and then append the new program. If the read is unsuccessful, read in the previous program again and exit to error. If successful, move the new program name into the user's table, and if this is a run only program, set the run-only bit, unless the program is in this user's own library. Call SEMIC, which sets up pointers for the language processor, dependent upon whether the program is uncompiled or semi-compiled.

9. Check if an abort was attempted during the previous steps, and if so, abort the user.
CHAIN (cont.)

10. If a line number was specified, search the program for the statement and, if found, put its absolute address into PRGCT. If no line number was specified, set PRGCT equal to SPROG. If the program is null, or if the line number cannot be found, clear the chain bit in the flags word. In any case, exit to SCHBL.
SAVE

The SAVE routine is called by a user to save a program in the library. Its operation is as follows:

1. Test for the existence of a program name and a non-null program.

2. If the user's program is in compiled form (CFLAG bit = 1), call DCMPL to put it into the form in which we will save it.

3. Check if the common area has been allocated. If not, call ALCOM, which computes the amount of space required for common. This is used to determine the start-of-program pointer which is saved in word 4 of the directory entry, a device which keeps the common area from being overwritten on GET's and CHAIN's.

4. Test to see that the user has sufficient disc space allocated to save the program. The test to be satisfied is:
   
   \[ \text{(disc currently in use)} + \text{(length of program in records)} \leq \text{(disc allowed)} \]

5. Search the next disc ADT for the first entry large enough to hold the program. Remember the address of the entry in SAVC.

6. Initiate a seek to the disc address at which program will be written.

7. Perform a directory search on the program to be saved. Fail if such an entry already exists.

8. If the directory track is full, call the SUPERSAVE routine to attempt to reallocate the directory. SUPERSAVE will perform step 9 itself and proceed to step 10.

9. Insert a new directory entry into the directory.

10. Update the IDT and disc ADT.

11. Copy the user's program to its library area. If the write is unsuccessful, set the "ill-stored" bit in the directory and fail.
CSAVE

The CSAVE routine is called by a user to save a program in semi-compiled form. This is the form it has after the symbol table is built. CSAVE operates like SAVE with the following exceptions:

2. and 3. If the user's program is in compiled form, call RSTPT, which restores the symbol table to its appearance just after it was built, and before variable storage has been allocated. If the program is in uncompiled form, call ALCOM and then jump into the compiler, returning after the symbol table is built.

9. Insert a new entry into the directory, setting bit 15 of word 3 to indicate a semi-compiled program.

11. Read the program and symbol table back from the drum. Prior to writing it to disc, append 7 words which are:

1) SYMTB - symbol table pointer
2) FILCT - -5 + # of FILES statements
3-6) FLSTS - pointers to FILES statements
7) USESN - "USING SEEN" flag
SUPERSAVE

The SUPERSAVE routine is called by the SAVE, CSAVE, COPY, BESTOW and OPEN routines when they want to make a directory entry on a track that is already full. SUPERSAVE assumes that the following words are set properly:

\[(\text{LTEMP}:\text{LTEMP+3}) = \text{first 4 words of entry.}\]
\[(\text{LTEMP+4}) = \text{pointer to DIREC entry for appropriate directory track}\]
\[(\text{LTEMP+5}) = \text{core address of entry which is to precede the new entry}\]
\[(\text{LTEMP+6: LTEMP+7}) = \text{disc address of entry}\]
\[(\text{LTEMP+8}) = \text{length of entry}\]
\[(\text{LTEMP+10}) = \text{start of program pointer/record size}\]
\[(\text{LTEMP+13}) = \text{drum address of entry}\]

Note that (LTEMP+4) and (LTEMP+5) are set correctly by DLOOK.

SUPERSAVE attempts to redistribute the directory tracks so that they will be as equal in length as possible. This will generally prevent it from being called very frequently. The operation is as follows:

1. Scan through DIREC and determine the total length of all directory tracks, and add 12 for the new entry. If all directory tracks are full, exit through failure location.

2. Divide total directory length by number of available disc tracks to determine their new individual lengths. Insert these in the table at (DEFNN+1:DEFNN+80) as negative.

3. Now squeeze all the directory entries to the lastmost of the available tracks. This is done by reading the tracks in reverse order and writing 8184 words on each track until we run out of directory entries. The following variables are used in this section:

   \(\text{(SUP)} \) \(K1\) points to the DIREC entry for track being read (initially DIREL)

   \(L1\) points to the DIREC entry for track being written (initially DIREL)

   \(K2 = \# \) of words in core
4. If \((\text{# of words on track } K1 - K2) > 8184\), go to 5. Otherwise update \(K2\) and read this track to the core buffer at location \(\text{LULEN} + K2\) (\(K2\) being negative). If \(K2 \neq -8184\), go to 8. Otherwise set the length into the \(L1\) DIREC entry and write the 8184 word buffer to track \(L1\). Set \(K2=0\) and go to 7.

5. Set \(ES=\left(\text{# of words on track } K1-K2-8184\right)/64\). That is the number of extra sectors on the track to be read. Set \(EX = ES \times 64\). This is the number of extra words. Then read the last \((\text{# of words on track } K1-EX)\) words from track \(K1\) to location \(\text{LULEN} + K2 - \text{# of words on track } K1 + EX\) and update \(K2\).

6. Write 8184 words to track \(L1\) from location \(\text{LULEN}-8184\) and set the length in the \(L1\) DIREC entry. Move the leftover \(-8184-K2\) words to the end of the buffer, resetting \(K2\). Then read the \(EX\) words left on track \(K1\) to location \(\text{LULEN}+K2-EX\). Set \(K2=K2-EX\).

7. Update \(L1\) to point to the next track to write.

8. Update \(K1\) to point to the next track to read. If we've finished all tracks, go to 9. Otherwise go to 4.

9. If \(K2 = 0\), go to 10. Otherwise write out the \(-K2\) words to track \(L1\) and set the length in the \(L1\) DIREC entry.

10. Zero out the lengths in the DIREC table of all those tracks that no longer have anything written on them.

11. Now redistribute the directory tracks. The basic idea of the algorithm is to fill the swap area with as much of the directory information as we can, reading from the beginning, and then to write out as much as we can, always making sure than when writing we don't overlay any portion that hasn't been read yet. The following variables are used:

(SUP) \(K1\) points to the DIREC entry for track being read
(initially DIREC0)

\(L1\) points to the DIREC entry for track being written
(initially DIREC0)

\(K2 = \text{# of words read so far from track } K1\)
(initially 0)
L2 = # of words written so far on track L1
   (initially 0)
P = # of words in core
   (initially 0)
PP points to DEFNN entry, telling how many are to be written
   on L1.
TG = 1 if we have already inserted the new entry.

12. If L2 = -(PP), we have completely written track L1 so check for
   L1 = DIREL. If it is, we've written all the tracks, so go to step 18.
   Otherwise, advance L1 to the next directory track advance PP, set L2 = 0,
   and repeat this step. If L2 = -(PP), go to step 13.

13. If P > 10232, we have read as much as we can, so go to step 15.
   If K1 = DIREU, there is nothing left to read, so go to step 15. If K2 = #
   of words on track K1, we've read the entire track, so advance K1 to the next
   track, set K2 = 0, and repeat this step. Otherwise, compute the number of
   words we can read. If there is room to read the balance of the track, we
   will, otherwise we will read the maximum number of full sectors possible.
   If this is zero, go to step 15. If it is not zero, read from sector K2/64
   into core location L1BUS +P. Add the number of words read to P and to K2.

14. If TG = 0, determine if we can insert the new entry. To do this
   we first determine where the even entry boundary occurs in the core buffer,
   since we may have read only part of an entry (12 does not divide 64 evenly).
   If the last entry in the buffer is greater than the entry we are inserting,
   the entry goes on this track. If this is not the case, go back to step 13.
   Otherwise, set TG to 1, make a 12-word hole, insert the new entry, set
   P = P + 12, and go back to step 13.

15. **Write section.** Set S = 0. This is the number of words written.
16. Compute number of words we can write on track L1. First set
   A = - number of words left to write on the track. If L1 = K1, we
   haven't finishing reading everything from track L1, so if L2-A > K2 change
   A to L2-K2, which is the number of words we can write without destroying
   any unread directory information. If P-S<-A, we don't have as much in core
   as we are capable of writing so set A = -(((S-P)+64) x 64), an exact number
   of sectors.
17. If \( A = 0 \), we can't write anything, so if \( S \neq D \) slide the remaining \( P-S \) words in core up to location \( \text{LIBUS} \), set \( S = 0 \) and \( P = P-S \). Then go back to step 12.

   If \( A \neq 0 \), write \(-A\) words to sector \( L2\div64 \) of track \( L1 \). If \( L2 = 0 \), set the first 4 words of the \( L1 \) \( \text{DIREC} \) entry to the first 4 words written. Set \( L2 \) to \( L2-A \), \( S \) to \( S-A \), and go back to step 16.

18. Set the new directory lengths into \( \text{DIREC} \) and go back to the calling program.
GET

The GET routine is called by a user to load a program from the library. The operation is as follows:

1. Translate name of program from user's input. If preceded by a "$", set up for A000 search; if preceded by a "#", set up for group library search; otherwise set for searching on user's id.

2. Perform directory search. Print error if not found.

3. Fail if entry is a file (Bit 15 of word 2 of entry is 1). Fail if entry is ill-stored (Bit 15 of word 4 of entry is 1).

4. If the program is on the disc, initiate a seek by calling the disc driver to perform a zero length transfer.

5. Check that the program will fit into the user area. This is necessary in case a program which was saved under an old version of the system can no longer fit with the current version.

6. Set the date into word 5 of the directory entry and write it back.

7. Read in the basic portion of the user area and the common area. Append the library program, reading it in starting with the word specified by the start of program pointer (word 4 of the directory entry). If the read is unsuccessful, read in the previous program again and fail. If successful, move the new program name into the user's table, and if this is a run only program, set the run-only bit unless the program is in this user's own library.

8. Call SEMIC, which sets up pointers as follows: For uncompiled programs, clear CFLAG bit and set SYMTB = 0. For semicompiled programs, set CFLAG bit, move 4 pointers to FILES statements into FLSTS, set FILCT, set SYMTB to point to the first word of the symbol table and set SPTR = 0. For both types of programs set MAIN to point to this user, set SPROG to the start of program pointer and set PBPTR to point past the last word used by the program.
APPEND

The APPEND routine is called by a user to append a library program onto his current program. The operation is the same as GET for steps 1-4, and then continues as follows:

4. Check that the program to be appended is not semicompiled and has no common area. Set the date into word 5 of the directory entry and write it back.

5. Load user's current program and call DCMPL. Check that the program to be appended will fit, and if so, read it in at the end of the current program. If the read is unsuccessful, fail.

6. If the current program is not null, search it for the sequence number of the last statement, and insist that it be smaller than the sequence number of the first statement of the appended program. If okay, update PBPTR and if the appended program is run-only, set the run-only bit unless the program was retrieved from the user's own library.
HEllo

The HELLO command is used to log a user on to the system. Its operation is as follows:

1. If the current ID is 0, there is no user to log off, so go to Step 2.
   Otherwise, clear the user's section of FLISS, and tell the I/O processor (service routine NUC) that a new user called. This will force the user to be disconnected if he does not successfully log on.

2. Read the IDT. If there is no user to be logged off, go to Step 4. Check if user has control over the line printer. If not, go to Step 3. Otherwise tell the I/O processor to disconnect the line printer from user (service routine LPD).

3. Find the old user's IDT entry and update his total time used. Add an entry to LOGGR to be printed on the system console. Set the user's ID word to 0.

4. Translate the new ID code and search for it in the IDT. If not found, print an error message and terminate. Compare the password typed to the correct one, and fail if they disagree.

5. Check if a terminal type was input. If not, assume terminal type #1 and go to Step 6. Otherwise check if terminal type is in the Range 1 through 6. If not, print an error message.

6. Tell I/O processor (service routine STP) which terminal is connected to the port. Check that the time used to date is less than the time allowed.

7. Add a LOGON entry to LOGGR and set the starting time into the user's table. Also insert the ID code, clear the name, clear the program and tell the I/O processor of successful Logon (service routine ULO).

8. Search the directory for a program named HELLO in the library of user Zggg. If not found, or if it is a file, or if it will not fit in core, or if it is ill-stored, or if it can not be read from drum or disc, print READY and terminate.
9. Read in the fixed user area and append HELLO. Call SEMIC, which sets pointers as in SAVE. Change the user's status to RUN, set TIMEF, and transfer to BASIC.
This command is used to log a user off. It operates as follows:

1. If user does not have control of the line printer, go to Step 2. Otherwise tell the I/O processor to disconnect the line printer from user. (system calls service routine "LPD" which sets the line printer disconnect flag LPDIS)

2. If user ID is 0, go to Step 3. Otherwise clear the user's FUSS table and read in the IDT. Compute the time used and update his IDT entry. Create a LOGOFF entry in LOGGR. Clear the user's ID entry and output a message.

3. Get ?TYPE from the I/O processor with service routine WTP. If ?TYPE = 0 (ASCII terminal connected to the port), tell I/O processor to restore this port to full duplex. Otherwise tell I/O processor to restore this port to half duplex.

4. Tell I/O processor to disconnect the user (service routine HUU) and then terminate.
The KILL routine is called by a user to delete a program or a file from the library. Files which are being accessed by another user are not allowed to be killed. The operation is as follows:

1. Translate the program or file name and perform a directory search. Fail if illegal name or the search fails.
2. If the entry is a file, search the FUSS table to see if any other user has access to the file. If so, print a message and terminate. If not, clear the user's section of FUSS.
3. Delete the entry from the directory and adjust DIREC. Subtract the program length from the user's IDT entry, and restore the space to the ADT (drum) if the entry was sanctified. Restore the space to the appropriate disc ADT.
4. If a file was killed, read the user's program in and decompile it. This guarantees that any old references to the file will disappear.
The function of RENUMBER is to assign a new set of sequence numbers to a user program. The user may specify the sequence number of the first statement and the increment between statements. If unspecified, these are set to 10. He may also specify the first statement to be renumbered and the last statement to be renumbered. If unspecified these are set to the first statement of the program and the last statement of the program respectively.

There are actually two sets of numbers that must be modified. One set is the sequence numbers themselves, each of which occupies the first word of its statement. The other is the set of references, which are labels in GO TO, GOSUB, RESTORE, PRINT USING, MAT PRINT USING, and IF statements. Each of these also occupies one word. For programs in compiled mode, they are pointers to the statement they reference; in decompiled mode they are the actual statement number.

The primary technique used is to change all the references to absolute pointers (if in decompiled mode), then to change all the sequence numbers, and then (if in decompiled mode) to change the references to the new statement numbers. References to nonexistent labels are left unchanged.

Because the process of changing all the references to absolute pointers can become quite time consuming (due to the search that must be performed for each reference), a table is built in advance essentially dividing the program into 32 parts, each containing the same number of statements. For large programs with many references, this effectively cuts the time down by a factor of close to 32.

The subroutine RENSK is used to scan for references. It maintains two pointers, P and Q. Whenever it is called, it moves P to the next reference, and sets Q to point at the statement following the one that P is pointing at. It takes advantage of the fact that any references within a statement are always the last word or words of the statement, except in the case of PRINT USING and MAT PRINT USING, in which case it takes advantage of the fact.
that there is only one statement number reference. Before calling RENSK for the first time, Q is set to point at the first statement to be renumbered. P is set to Q-1.

The operation of RENUMBER is as follows:

1. If null program, terminate immediately. Otherwise, read in user program.
2. Translate and check parameters M and N.
3. Translate parameters P and Q. Set RENBA = first statement to be renumbered, RENLA to last statement to be renumbered.
4. Set RENLA to point to the last sequence # ≤ RENLA. Also set RENBA to point to the first sequence # ≥ RENBA.
5. Insure that there will be no sequence number overlap at either end of the portion of the program to be renumbered and that the new sequence numbers will not exceed 9999.
6. If program is in compiled mode, go to step 9. Otherwise, set up a table in ERSEC which divides the program into 32 parts. The result is that for each I from 0 to 31
   ERSEC [I] = sequence number of first statement in part I,
   ERSEC [I+32] = absolute address of that statement
   If there are 32K + L statements (0 ≤ L ≤ 31) in the program, ERSEC [I] is the sequence number of statements:
   \[
   \begin{align*}
   (K + \ell) I + 1, & \quad \text{if } I < L \\
   KI + L + 1, & \quad \text{if } I \geq L, K > 0 \\
   L & \quad \text{if } I \geq L, K = 0
   \end{align*}
   \]
   Set Q = SPROG, P = Q-1. (SPROG points to the first statement).
7. Call RENSK to find the next statement reference. If there are none left, go to step 9. Find the largest I for which ERSEC [I] < (RENP). If there is none, the statement referenced does not exist, so go to step 8. Otherwise, test all statements from(ERSEC [I + 32]) to either (ERSEC [I + 33]) or PBPTR, depending upon whether I < 31 or I = 31. If found, set (RENP) to the location of the statement referred to, and repeat this step. Otherwise, go to step 8.
8. Set (RENP) = (RENP) + 1000000 and go back to step 7.
9. Change the sequence numbers of all statements to be changed, according to the \( M, N, B, A \) and \( L \) parameters. If compiled mode, terminate. Otherwise, set \( Q = \text{SPROG}, P = Q-1 \), and go to step 10.

10. Call \text{RENSK} to find the next statement reference. If none left, terminate. If \((\text{RENP}) < 0\), the reference was undefined, so set \((\text{RENP}) = (\text{RENP})-100000_8\), and repeat this step. Otherwise, set \( \text{RENP} = \left( (\text{RENP}) \right) \) and repeat this step.

\[
\begin{align*}
N &= 48 \\
K &= 1 \\
L &= 16 \\
\text{ER2SEC}[i] &= 2 \cdot i + 1 \quad (i < 16) \\
\text{ER2SEC}[i] &= i + 16 + 1 \quad (i \geq 16) \\
&= 1, 3, 5, \ldots, 31 \\
&= 33, 39, \ldots, 48
\end{align*}
\]
NAME

The NAME routine is called by a user when he wants to assign a name to his program. The program name is placed in his teletype table. The operation is as follows:

1. Get an input character. If a carriage return change it to a blank. If a control character, ignore it and repeat this step. If a "$" or "*", and this is the first character, print an error message and terminate. If a ",", print an error message and terminate.

2. Add the character to the user's name area. If <6 characters, go back to step 1. Otherwise, restore the RUN-ONLY bit, and get one more character. If not a blank, print an error message. Then terminate.
CATALOG

The CATALOG routine prints a list of all programs and files in the user library. The operation is as follows:

1. Output heading line.
2. Perform directory search on the program with all nulls. Get first directory entry following the one sought.
3. If the entry does not belong to this user, output a CRLF and terminate. Otherwise, output the 6 characters of the name one at a time, then a blank, then a 'C' if a semi-compiled program or an 'F' if a file (or a blank if neither), then a 'P' if the entry is protected (otherwise, a blank), then a 'S' if the entry is sanctified (otherwise a blank), then the 5-digit program or file length with leading zeroes suppressed, then 3 blanks. Program length is printed in words (stored as negative number) and file length in records.
4. If <4 names have been printed on the line, advance to the next directory entry and return to step 3. Otherwise, copy the name of the last one output into the user's teletype table, output a carriage return and suspend until the buffer is almost empty.
5. Read the name of the last program printed from the teletype table and perform a directory search. The reason for doing this in this way rather than saving a pointer to the directory is that during the time CATALOG was suspended, the directory may have been changed in any way. Get the first directory entry following and go back to step 3.
LIBRARY

The LIBRARY routine prints a list of all programs and files in the public library. Its operation is identical to that of CATALOG except that A0000 is used for directory searches instead of the user's id, and ill-stored programs are not listed.

GROUP

The GROUP routine prints a list of all programs and files in the user's group library (the library of the idcode ending in 00 which has the same letter and first number as the user). Its operation is identical to that of LIBRARY except that the group librarian's idcode is used for directory searches instead of A0000.
The DIRECTORY routine prints a list of all directory entries on the user console. The entries are printed one per line, and consist of id, name, last reference date, length, disc address and drum address, if any. The operation is as follows:

1. Check that the user's id is A000. If not, fail.
2. Check to see if an idcode was specified. If so, we will start printing the directory with this idcode.
3. Print the heading consisting of system id, date and time and suspend.
4. Print the directory heading and suspend.
5. Set up parameters for directory search for null program name and idcode previously determined (or null idcode if none specified).
6. Perform directory search.
7. Get first directory entry following the one sought. If pseudo-entry, terminate.
8. If id of entry is different from that of the preceding entry, output the ASCII representation of the idcode. Otherwise output four blanks. Save the idcode in the RTIM word of the user's TTY table.
9. Output the six character program name and save it in the TEMP words of the user's TTY table.
10. Output the last reference date.
11. Output a 'C' for semi-compiled programs, an 'F' for files, and/or a 'P' for protected programs or files.
12. Output the length, the disc address, and the drum address, if there is one.
14. Retrieve the parameters for the directory search from the user's TTY table and go to step 6.
**SDIRECTORY - USER CONSOLE**

The SDIRECTORY routine prints a list of all sanctified programs and files on the user console. The printout is in the same format as a DIRECTORY printout. The routine functions the same as DIRECTORY, except that in step 7 a check is made to see if the entry is sanctified. If it is, the processing continues as in DIRECTORY. Otherwise, the pointer is moved to the next entry and step 7 is repeated.
REPORT - USER CONSOLE

The REPORT command prints IDT information on the user console. For each IDT entry, the user ID, time consumed, and disc consumed are printed. The entries are printed three per line. Note that the time printed on the console does not include any time for currently active users, since these are not added to the IDT until the user logs off. The operation of REPORT is as follows:

1. Check that the user's id is A000. If not, fail.
2. Check to see if an idcode was specified. If so, save it, as we will start printing the report with this idcode. Otherwise save a null idcode.
3. Print the heading consisting of system id, date and time and suspend.
4. Print the report heading and suspend.
5. Retrieve the idcode and find what track it's on. Read that track and locate the idcode.
6. Output the id, time and disc of the next three entries. If necessary, read the next id track.
7. If no entries left, print XOFF,CR,LF,LF and terminate. Otherwise save the present idcode + 1, print XOFF,CR,LF and suspend. Go to step 5.
STATUS - USER CONSOLE

The STATUS routine prints a summary of the various system resources and the extent of their utilization on the user console. It operates as follows:

1. Check that the user's id is A000. If not, fail.
2. Print the heading consisting of system id, date and time and suspend.
3. Print MAGSC and a '*' if the mag tape unit is a 7970 (bit 15 of MAGSC=1). Print the select codes of the 4 drums.
4. Print the logical unit, select code, unit number, first block and last block of the discs on the system.
5. Print a list of those tracks which are locked on each drum.
6. Print a list of those disc blocks which have been MLOCKED.
7. Set the ?STAT word in the user's TTY table to the status overlay and suspend so that when we come back the overlay will be read in.
8. Get the line printer select code and type from the I/O processor and print them ('*' if the printer is a 2610A, '**' if the printer is a 2767A). Then print the port number of the current user.
9. Print the drum addresses and lengths of the IDT, ADT, Disc ADT and Directory tracks.
10. Print the drum addresses of the system library tracks and user swap tracks.
11. Print the disc addresses of the 32-block areas reserved for the IDT, Disc ADT and Directory.
12. Print the disc addresses and lengths of the system segments.
13. Terminate.
DELETE

The DELETE command allows a user to delete a section of his program. He can specify two parameters, M and N. M refers to the first line to be deleted, N to the last. If N is not specified, the entire program is deleted, starting at line M. The operation is as follows:

1. Translate and check parameters. If N is not specified, set it to 9999.
2. Decompile program.
3. Locate range of statements to be deleted.
4. Move portion of program following deleted area up against portion preceding.
5. Reset PBPTR and exit.
TIME

The TIME command prints the user's console time and total time. The operation is as follows:

1. Print "CONSOLE TIME ="
2. Read IDT.
3. Compute console time and print it.
4. Print "TOTAL TIME ="
5. Find user's IDT entry. Add the time in there to the console time and print it.
The PROTECT command allows user A000 or any group librarian to protect a program or file. Program protection means that no other user may list or save the program. File protection means that no other user may access the file. Files are always protected against other users writing on them. The operation is as follows:

1. Check for privileged user.
2. Translate and check the program or file name.
3. Perform a directory search on the specified program. Fail if not found.
4. Set the protect bit (BIT 15 of word 1 of the directory entry), write the directory back to the disc, and terminate.
UNPROTECT

This is identical to PROTECT except that it clears the protect bit.
OPEN

The OPEN command is used to create data files. The user must specify the file name and file length in records. He may also specify a logical record size. The operation of OPEN is as follows:

1. Translate and check the file name and length and record size.
   File names are subject to the same restrictions as program names. File length must be between 1 and 32767 records, inclusive. Record size, if specified, must be between 64 and 256, inclusive. (The default record size is 256 words.)

2. Check the IDT and disc ADT's to see if a) the user has enough disc space allocated to him to satisfy the command; and b) there is an area on the disc which is large enough to accommodate the file. Save the location of the disc ADT entry and its information, but don't update it until we know there is room in the directory.

3. Perform a directory search on the file name. If found, this is a duplicate entry, so terminate. Otherwise, if the directory track is not full, insert the new entry. If it is full, call in SUPERSAVE to restructure the directory and insert the entry.

4. Update the IDT and disc ADT appropriately.

5. Fill the user area with end-of-file marks (a -1: in the first word of each of 40 256-word blocks). Write this area to the location on the disc reserved for the file. Increment the disc address by 40 blocks and write another 40 records up to 10 times (total) or until the file is full. The last write may be from 1 to 40 blocks in length.

6. If the file has been filled, terminate. If not, save the low word of the disc a-dress immediately beyond the last write in the user's teletype table, along with the file name. Move the user to the bottom of the queue and suspend.
7. Retrieve the file name and partial disc address from the
teletype table. Ascertain that the file exists (is in the
directory) and that the reconstructed disc address falls with­
in the file. If not, terminate. Otherwise, return to step 5.
LENGTH

The LENGTH command prints the length of the user's program, as it would be if saved. This is only the length of the source area of the program, and includes neither the fixed portion nor any of the tables used at run time. The length is determined in one of two ways:

1. If the user is in decompiled mode, length = PROG-SPROG. PROG is just a copy of PBPTR, which points to the last word +1 of the program. PBUFF points to the first word.
2. If the user is in compiled mode, length = SYMTB-SPROG.
**ECHO**

The ECHO command is used to control the computer echo of teletype input. Echoing is determined by the user's bit in the word PLEX or PLEXI in the I/O processor. Bit = 0 implies no echo, 1 implies echo. The user will want echoing if any only if his teletype is full duplex. The command format is:

- **ECHO-ON** for full duplex.
- **ECHO-OFF** for half duplex.
MESSAGE

The MESSAGE command is used to send a message from a user console to the system console. The message is placed in a queue and is ultimately output to the system console by the scheduler. The routine operates as follows:

1. Check if message queue is full. If so, fail.
2. Put a CR-LF and the ASCII representation of the user's port number in the message buffer.
3. Move the message from the user's teletype buffer in the I/O processor to the message buffer.
4. Increment message counter and set pointer to next message buffer.
5. Terminate.
The LPRINTER command is used to obtain the line printer as the output device. The routine operates as follows:

1. Check to see if the line printer is currently being used. If so print "LP BUSY".
2. Ask the I/O processor to check on the availability of the line printer. The I/O processor will return the following status in the A register:

   A ≠ 0   Line printer available and on-line. The line printer is assigned to the user and the character string following the command word (if present) will be printed.

   A = 0   Line printer is not available or not on line. Print "LP NOT AVAILABLE".

   A < 0   Line printer available but character string is too long. Print "ILLEGAL FORMAT".

3. Set the LPRINTER command flag, LFLAG, and place the address of the user's port number in the LP user indicator PRIST. Upon completion of the user's next command, PRIST will be cleared, the line printer will be released, and a completion message will be sent to the user. LFLAG is used to indicate that command being terminated is LPRINTER and therefore the line printer is not to be released.
4. Terminate. LFLAG is cleared in the termination routine.
The REPORT command prints IDT information on the system console. From each IDT entry, the user id, time consumed, and disc consumed are printed. The entries are printed three per line. Note that the time printed on the console does not include any time for currently active users, since these are not added to the IDT until the user logs off. The operation of REPORT is as follows:

1. Check to see if an idcode was specified. If so, save it, as we will start printing the report with this idcode. Otherwise save a null idcode.
2. Print the heading consisting of system id, date and time and suspend.
3. Print the report heading and suspend.
4. Retrieve the idcode and find what track it's on. Read that track and locate the idcode.
5. Output the id, time and disc of the next three entries to the buffer. If necessary, read the next id track.
6. If no entries left, print the buffer and terminate. Otherwise save the present idcode +1, print the buffer and suspend. Go to step 4.
DIRECTORY - SYSTEM CONSOLE

The DIRECTORY routine prints a list of all directory entries on the system console. The entries are printed one per line, and consist of id, name, last reference date, length, disc address and drum address, if any. The operation is as follows:

1. Check to see if an idcode was specified. If so, we will start printing the directory with this idcode.
2. Print the heading consisting of system id, date and time and suspend.
3. Print the directory heading and suspend.
4. Set up parameters for directory search for null program name and idcode previously determined (or null idcode if none specified).
5. Perform directory search.
6. Get first directory entry following the one sought. If pseudo entry, terminate.
7. If id of entry is different from that of the preceding entry, place the ASCII representation of the idcode in the output buffer. Otherwise, place blanks in the buffer. Save the idcode in location 35 of the buffer.
8. Move the 6-character program name to the buffer.
9. Convert the last reference date and put it in the buffer.
10. Convert the drum address and put it in the buffer, unless it is zero.
11. Convert the disc address and length and put them in the buffer.
12. Put a 'C' for semi-compiled programs, an 'F' for files, and/or a 'P' for protected programs or files in the buffer.
13. Print the line and suspend.
14. Set up parameters for directory search. These can be gotten from locations 35, 3, 4, and 5 of the buffer. Go to step 5.
SDIRECTORY - SYSTEM CONSOLE

The SDIRECTORY routine prints a list of all sanctified programs and files on the system console. The printout is in the same format as a DIRECTORY printout. The routine functions the same as DIRECTORY, except that in step 6 a check is made to see if the entry is sanctified. If it is, the processing continues as in DIRECTORY. Otherwise, the pointer is moved to the next entry and step 6 is repeated.
STATUS - SYSTEM CONSOLE

The STATUS routine prints a summary of the various system resources and the extent of their utilization on the system console. It operates as follows:

1. Print the heading consisting of system id, date and time and suspend.
2. Print MAGSC and a '*' if the mag tape unit is a 7970 (bit 15 of MAGSC=1). Print the select codes of the 4 drums.
3. Print the logical unit, select code, unit number, first block and last block of the discs on the system.
4. Print a list of those tracks which are locked on each drum.
5. Print a list of those disc blocks which have been MLOCKED.
6. Get the line printer select code and type from the I/O processor and print them ('*' if the printer is a 2610A, '**' if the printer is a 2767A). Then print the port number of the current user.
7. Set the console status to the status overlay and suspend so that when we come back the overlay will read in.
8. Print the drum addresses and lengths of the IDT, ADT, Disc ADT and Directory tracks.
9. Print the drum addresses of the system library tracks and user swap tracks.
10. Print the disc addresses of the 32-block areas reserved for the IDT, Disc ADT and Directory.
11. Print the disc addresses and lengths of the system segments.
12. Terminate.
ROSTER

The ROSTER routine prints a listing of the Id codes of all active users. These are obtained from the ID word in the 32 TTYTABLES. The absence of a user is indicated by the word being zero.
The ANNOUNCE command is used to send a message from the system console to any or all of the user consoles. It operates as follows:

1. Get the port number to send the message to. If 'ALL' specified, set up for sending message to all ports.

2. Output CR-LF-LF, followed by the message, followed by CR-LF-LF to a port. This output is done a character a time, after insuring that the I/O processor can take the character without overflowing the buffer.

3. Move to the next port, and if there are any more ports to do, go to step 2. Otherwise terminate.

NOTE: If a user has the line printer as his output device, the announce message is not sent to him. The flag PRIST indicates the address of the TTY# of the current user.
RESET

The RESET command modifies the time to date of a user's IDT entry. It operates as follows:

1. Set ID = T = 0.
2. If the idcode = "ALL" go to 3, otherwise set ID = the specified ID code.
3. If no time specified, go to 4. Otherwise set T = specified time.
4. Read the IDT track for ID. If ID = 0, go to 5. Otherwise search for the specified idcode. Fail if not found. If found, set its time entry to T, write the IDT track back and terminate.
5. Set the time entry for all the idcodes on this track to T and write it back to drum.
6. Move to the next IDT track. If all are finished, terminate. Otherwise read the IDT track and go to 5.
CHANGEID

The CHANGEID command is used to modify any or all of the parameters in an IDT entry. The parameters that can be specified are: password, time allowed, disc allowed. The operation is as follows:

1. Translate id specified. Read IDT track for this id and locate the specified id. Fail is not found.
2. If password specified, insert into IDT entry. If followed by comma, go to step 3, otherwise to step 5.
3. If time specified, insert into entry. If followed by comma, go to step 4, otherwise to step 5.
4. Insert new disc value.
5. Write IDT track back to drum and terminate.
SLEEP

The SLEEP command is used for system shutdown. It operates as follows:

1. Remove all users from the queue and make sure they can't get back by:
   a) clearing each user's ?FLAG word in his TTY table.
   b) setting all status words to -2.
   c) setting T35LK to point to MLINK+1.

2. Output the sleep message to all active users, preceded and followed by a CRLF.

3. Tell the I/O processor to disconnect the telephones.

4. Call LCD to update the last change date for files for each port that has a program that is still active.

5. Update the IDT entry for each active user and create a logoff entry in LOGGR.

6. Wait for the console to finish outputting.

7. Read in the loader, turn off all the I/O and the interrupt system, set power fail to halt.

8. Set A = 0 (sleep) and jump to the dump.
The HIBERNATE command is identical to the SLEEP command except for the following additions/changes:

0. If MAGSC = 0, fail. Otherwise set the current time into HDATE.
8. Set A = -1 (hibernate) and jump to the dump.
NEWID

The NEWID routine adds on entry to the IDT. The operation is as follows:

1. Translate the idcode.
2. Determine what IDT track the idcode is on and read it in.
3. Translate the other parameters.
4. Search the IDT for the specified id. Fail if found. If the track is full go to 5. Otherwise insert the new entry in its appropriate position, update the track length, write the IDT track back to drum and terminate.
5. Scan through IDEC, determine the total length of all IDT tracks, and add 8 for the new entry. If all tracks are full, fail.
6. Divide the total IDT length by the number of IDT tracks to determine their new individual lengths. Insert these in the table at (NNSNN+1:NWSNN+3) as negative.
7. Now redistribute the IDT tracks. The basic idea of the algorithm is to fill the swap area with as much of the IDT information as we can, reading from the beginning, and then to write out as much as we can, always making sure that when writing we don't overlay any portion that hasn't been read yet. The following variables are used:
   - K1 points to the IDEC entry for track being read
   - L1 points to the IDEC entry for track being written
   - K2 = # of words read so far from track K1 (initially 0)
   - L2 = # of words written so far on track L1 (initially 0)
   - S = # of words in core (initially 0)
   - SP points to NWSNN entry, telling how many are to be written on L1.
   - TG = 1 if we have already inserted the new entry.
8. If L2 = (SP), we have completely written track L1 so check for L1 = NI DC2. If it is, we've written all the tracks, so go to step 14.
   Otherwise, advance L1 to the next directory track, advance SP, set L2 = 0, and repeat this step. If L2 - (SP), go to step 9.
9. If \( S > 10232 \), we have read as much as we can, so go to step 11. If \( K_1 = \text{NIDC3} \), there is nothing left to read, so go to step 11. If \( K_2 = \text{# of words on track } K_1 \), we've read the entire track, so advance \( K_1 \) to the next track, set \( K_2 = 0 \), and repeat this step. Otherwise, compute the number of words we can read. If there is room to read the balance of the track, we will, otherwise, we will read the maximum number of full sectors possible. If this is zero, go to step 11. If it is not zero, read from section \( K_2/64 \) into core location \( \text{LIBUS} + S \). Add the number of words read to \( S \) and to \( K_2 \).

10. If \( TG = 0 \), determine if we can insert the new entry. This will be so if \( K_1 = \text{IDLNP} \) and \( V - \text{LIBD} < K_2 \). If this is not the case, go back to step 9. Otherwise, set \( TG \) to 1 and insert the new entry in core. Set \( S \) to \( S + 8 \) and go back to step 9.

11. Write Section. Set \( SS = 0 \). This is the number of words written.

12. Compute number of words we can write on track \( L_1 \). First set \( A = -\text{number of words left to write on the track} \). If \( L_1 = K_1 \), we haven't finished reading everything from track \( L_1 \), so if \( L_2 - A > K_2 \) change \( A \) to \( L_2 - K_2 \), which is the number of words we can write without destroying any unread IDT information. If \( S - SS < -A \), we don't have as much in core as we are capable of writing, so set \( A = (((SS - S) \div 64) \times 64) \), an exact number of sectors.

13. If \( A = 0 \), we can't write anything, so if \( SS \neq 0 \) slide the remaining \( S - SS \) words in core up to location \( \text{LIBUS} \), set \( SS = 0 \) and \( S = SS - S \). Then go back to step 8. If \( A \neq 0 \), write \(-A\) words to sector \( L_2 \div 64 \) of track \( L_1 \). If \( L_2 = 0 \), set the first word of the \( L_1 \) IDEC entry to the first word written. Set \( L_2 \) to \( L_2 - A \), \( SS \) to \( SS - A \), and go back to step 12.

14. Set the new IDT lengths into IDEC and terminate.
KILLID

The KILLID routine removes a specified id from the system. The operation is as follows:

1. Get the id. If the id is A000, or if it ends in '00' and any members of that group are logged on, fail. This is because the files belonging to A000 and group librarians may be accessed by other users, and removing them would be almost impossible.

2. Read the IDT track for the specified id and search it for the id. Fail if not found. Otherwise, delete the entry from the IDT and write it back to the drum.

3. If any user with the specified id is currently on the system, set the id item of this TTY table to 0, set his status to -2, set his COM14 bit to force him to be disconnected, and remove him from the queue if he is on it. Also, zero out his section of the FUSS table.

4. Remove all directory entries belonging to this user and build a table which will be used to patch the ADT and Disc ADT. For each directory entry, four words are placed in the table: drum address, length, and disc address.

5. Write the directory back to disc. Read the ADT, call RSFS to return the space released from sanctified programs and files, and write the ADT to drum.

6. Call TBDAD to return released space to the Disc ADT.

7. Terminate.
UNLOCK

The UNLOCK command is used to restore drum tracks to the system. The operation is as follows:

1. Interpret parameters, setting F and L to the first and last tracks to be unlocked.
2. Scan the TRAX table to determine the number of tracks to be unlocked. Set CN to this number.
3. Set CN = min(CN, (8192 + ADLEN)/2). The parenthesized expression is the number of words that can be added to the ADT.
4. Read the ADT into core location LIBUS + 2 CN.
5. Set MOVED = LIBD, MOVES = LIBD + 2CN.
6. If track F is unlocked go to step 8. Otherwise, unlock it by clearing its bit in TRAX. If MOVES is the ADT entry, so go to step 8.
8. If F ≠ L, set F = F + 1 and go to step 6. Set ADLEN = ADLEN - 2CN. Write the ADT back to drum and terminate.
The LOCK routine is used to tell the system that certain drum tracks are not to be used. Only tracks which are part of the program library are lockable, but tracks which contain active files are not. Any programs or files on tracks being locked are removed from the system. The operation is as follows:

1. Interpret the parameters and set F and L to the first and last tracks to be locked. Check that none of these tracks is used for swapping, directory, Disc ADT, IDT, ADT, or system. Fail if they are.

2. Search the directory for sanctified entries on the specified tracks. For each such entry, add a 4-word entry to the patch table consisting of a pointer to the Direc entry, a pointer to the entry in the buffer, and the two-word disc address. There is room for 512 entries in this table. Fail if it overflows.

3. Compare the patch table to the FUSS table. Fail if any of the entries in the patch table are active files.

4. Read the ADT, delete from it all entries for the tracks to be unlocked, and write it back.

5. For each track to be locked, set its TRAX bit to 1.

6. Update the directory using the patch table. For programs, set the drum address in the directory entry to 0. Also set the second word of the patch table entry to 0. For files, remove the entry from the directory. Set the first 2 words of the patch table entry to the id and length of the file.

7. Call TBDAO to return disc space formerly used by removed files to the Disc ADT.

8. Call TBIDT to update the space used for users whose files have been removed due to the locking.

MUNLOCK

The MUNLOCK command is used to restore to the system any disc blocks which have previously been MLOCKed. It operates as follows:

1. Interpret the parameters. Fail if the last block is less than the first block, or one of the blocks specified is one of the first 4 blocks of a disc, or the first and last blocks are not on the same disc, or the blocks are on a non-existent disc.

2. Read the Bad Blocks Table and Disc ADT for the disc specified.

3. Search the Bad Blocks Table for the first entry greater than or equal to the first block to be unlocked. If an equal entry is found check to see if the locked block is completely enclosed. If not, update the address and length of the locked block entry, return the unlocked space to the Disc ADT and go to 6. If so, eliminate the completely enclosed entry, return its space to the Disc ADT, and go to 5.

4. If a greater entry was found or if the end of the table was reached, check the preceding block to see if a portion of it is to be unlocked. If so, modify the entry. If the unlocked portion is in the middle of the block, it will be necessary to make a new entry in the table in addition to modifying the entry. After so doing, return the freed space to the Disc ADT. If the end of the table was reached in step 3, go to 6.

5. Check the next block to see if any portion of it is to be unlocked. If not, go to 6. If it is completely enclosed, eliminate it, return the space to the Disc ADT, and repeat step 5. If only part of it is to be unlocked, return this space to the Disc ADT and modify the address and length.

6. Write the Bad Blocks Table back to the disc and the Disc ADT back to the drum.

7. Terminate.
MLOCK

The MLOCK command is used to make certain disc blocks unavailable to the system because they are faulty or for some other reason. It operates as follows:

1. Interpret the parameters. Fail if the last block is less than the first block, or one of the blocks specified is one of the first 4 blocks of a disc, or the first and last blocks are not on the same disc, or the blocks are on a non-existent disc.

2. Read this System Segment Table. Fail if any of the specified blocks are reserved for system segments.

3. Read the Disc Allocation Table. Fail if any of the specified blocks are reserved for system usage as specified in the DAT.

4. Search the directory for programs or files whose disc address lie in the range of the blocks to be locked. For each such entry found, add a 4-word entry to a table consisting of a pointer to the Direc entry for this track, a pointer to this entry position in the directory buffer, and the two-word disc address. If an entry is found which is partially contained in the area to be locked, put its disc address and length in a special table. There may be 2 such entries. There is room for 512 entries in the main table. If it overflows, fail.

5. Read the FUSS table. Compare the table just built with the FUSS to determine if any of the blocks to be locked contain active files. If so, fail.

6. Read the MLOCK overlay.

7. Read the Bad Blocks Table. Then search it for the first entry greater than or equal to the first block to be locked. If an equal entry is found, and the new block is longer, reset the length and go to 8. If it is not longer, terminate. If an equal entry was not found, check the preceding entry and if the blocks to be locked will make it longer, update its length. If the table was full and the entries were
not combined, fail. If they were combined, go to 9. If the entry was to go at the end of the table and they were not combined, insert the new entry. Then go to 9. If a greater entry was found and the entries were combined, go to 8. If they were not, insert the new entry (unless the table was full, in which case we fail).

8. Check the next entry to see if it is overlapped by the one we just made. If not, go to 9. If so, eliminate the overlapped entry, and change the length of the new entry if it is longer. Repeat step 8.

9. Write out Bad Blocks Table.

10. If there were no entries in the table of directory entries to be removed, go to 13. Otherwise set TP to point to the first entry in this table. Read the directory and set MOVES=MOVED=LIBD.

11. MEM[TP + 1] = > the disc address of the directory entry. Replace MEM[TP] with the id of the entry, MEM[TP + 1] with the length in blocks, MEM[TP + 2] with the drum address, and, if the drum address is not 0, MEM[TP + 3] with the length in sectors. Call the move routine. Set MOVES=MOVES+12. This eliminates the directory entry we were pointing to. Set TP=TP + 4. Bump CT. If CT=0, go to 12. If MEM[TP]=DI, the next entry is on the same track, so go repeat 11. Otherwise move the end of the directory, write out this directory track, read the new directory track, set MOVES=MOVED=LIBD and repeat 11.

12. Read the ADT, call RSFS for each non-zero drum address entry in the table to return space to the ADT, and write the ADT.

13. Read the Disc ADT for this disc. For each of the two special table entries, if they exist call RADT to return their space. This is done because a locked block may remove only part of a program or file, and it is necessary to put the rest of the space back in the Disc ADT.

14. Remove from the Disc ADT any blocks which lie in the range of the blocks we are locking. Then write the Disc ADT back to drum.

15. If there are no entries in the main table, go to 16. Otherwise call TB1DT, which returns space to the IDT based on the first two words of each 4-word entry in the table we built.

16. Terminate.
The COPY command is used to copy a program or file from one user's library to the library of another user. It operates as follows:

1. Interpret the parameters.
2. Search the directory for the old entry. Fail if not found.
3. Save the file flag, semi-compiled flag, word 4, drum address, disc address and length.
4. Find out which IDT track the new idcode is on, read it, and search for the new idcode. Fail if not found. Also fail if there is not enough space in the user's library.
5. Search the Disc ADT's for enough disc space to put the new program. Fail if not enough space left.
6. Search the directory for the new entry. Fail if found. If the track is full, call SUPERSAVE and go to 8.
7. Add the new entry to the directory and write the directory track back.
8. Read the IDT, update the space used, and write it back.
9. Read the Disc ADT, update the space used, and write it back.
10. If the old program or file is on the drum, read it from the drum, write it to the new entry's place on the disc and terminate. Otherwise, read it from the disc and write it to the new entry's place on the disc 40 blocks at a time until it is completely copied. Then terminate.
The BESTOW command is used to transfer programs or files from one user's library to another user's library. It operates as follows:

1. Interpret the parameters. If no name is given, set name to null and go to 3.
2. Search the directory for the named entry in the old user's library. Fail if not found. Otherwise go to 4.
3. Search the directory for an entry in the old user's library. If one found, go to 4. If none found and this is the first time thru this step, fail. Otherwise print a message if there were any duplicate entries and terminate.
4. Save pointers to the directory entry and position in the directory buffers of this entry and also save its length.
5. Search the directory for an entry in the new user's library with this name. If found, bump the duplicate entry counter and go to 11.
6. Read the IDT track for the new user and insure that there is enough space in his library for this entry. If not, fail. Otherwise, update his disc space used and write out the IDT.
7. Read the IDT track for the old user, reduce his disc space used and write out the IDT.
8. Read the directory track for the old entry again. Save pertinent information, eliminate that entry from the directory, and write out the directory track.
9. Search the directory for a place to put the new entry. If the track is full, call SUPERSAVE and go to 11.
10. Insert the new entry in the directory and write the directory track back to drum.
11. Increment the name so we won't find the same entry again. If we were only transferring one entry, terminate. Otherwise, go to 3.
The SANCTIFY command is used to copy a program or file from the disc to the drum. The space on the disc is reserved so that it may be copied back at sleep time. The routine operates as follows:

1. Interpret the parameters.
2. Search the directory for the named entry. Fail if not found. Also fail if the entry is longer than 32 blocks or if the entry is already sanctified.
3. If the entry is a program, go to 4. Otherwise read the FUSS table and search it for this entry. Fail if found.
4. Read the ADT. Search it for an area large enough to put this entry. Fail if not found. Otherwise, update the ADT and write it back.
5. Read the directory track for this entry again, update the drum address for this entry, and write the directory track back.
6. Read the program or file from disc and write it to the drum.
7. Terminate.
DESECRATE

The DESECRATE command is used to return a sanctified program or file to its area on the disc. Programs are not copied back because there is an identical version already on the disc. The routine operates as follows:

1. Interpret the parameters.
2. Search the directory for the named entry. Fail if not found. Also fail if the entry is not sanctified.
3. If the entry is a program, zero its drum address, write the directory track back, and go to 6. Otherwise read the FUSS table and search it for this entry. Fail if found.
4. Zero the drum address of the file and write the directory track back.
5. Read the file from drum and write it to the disc.
6. Read the ADT, call RSFS to return the drum space to it, and write the ADT back.
7. Terminate.
The PURGE routine is used to delete from the library all programs or files which have not been referenced since a certain date. The operation is as follows:

1. If HELLO program exists, assign it today's date. This is because the HELLO routine does not perform this function.
2. Interpret parameters and set DT to the purge date. Make sure that DT < today's date.
3. Make sure that FUSS is empty. This is to avoid killing any active files.
4. Read a directory track. Set P = MOVED = MOVES = LIBD. Set ND to point to the end of the directory.
5. Test the entry pointed to by P to see if it should be deleted. If not, go to 7. Otherwise add a 5-word entry to the patch table consisting of id, length, disc address and drum address. Call the move routine and set MOVES = MOVES + 12.
6. If the patch table is full, write out the interim directory, call PURFX, and read back the directory.
7. Set P = P + 12. If P = ND, we are finished with this directory track, so go to 8. Otherwise to to 5.
8. Call the move routine to move the end of the directory track. Write out the track and move to the next track. If all tracks have been read, call PURFX and terminate. Otherwise go to 4.
The PURFX routine uses the patch table to update the ADT, Disc ADT and IDT as follows:

1. Read the ADT. Set MOVES = MOVED = L8192. Examine each entry for a non-zero drum address, and if one is found call RSFS to return space to the ADT. In any case call the move routine after examining the entry to delete the drum address (i.e., make it into a 4-word entry table). After returning all drum space write the ADT back.

2. Call TBDAD to return disc space to the Disc ADT.

3. Call TBIDT to adjust the disc space used in the IDT for each user who has lost programs or files.
MAGTAPE

The MAGTAPE routine is used to set a select code into the location MAGSC. Typing a '*' after the select code indicates that the tape unit is a 7970 and will force bit 15 of MAGSC to be set.
The PHONES command is used to tell the I/O processor how long to allow the user to try to successfully log on before disconnecting him. It is originally assumed to be 120 seconds. It can be reset to from 1 to 255 seconds by the PHONES command.
The PRINTER command is used to tell the I/O processor the select code of the line printer and the line printer type. '**' after the SC indicates a 2610A and '***' indicates a 2767A.
The SPEED routine is used to configure the specified port(s) at the specified baud rate and character size. The baud rate to be input is computed with the formula:

\[
\frac{14,400}{\text{Bit Rate}} - 1
\]

Where bit rate = # of Chars. per second X # of bits per character including the start and stop bit(s).

Note: If \( \frac{14,400}{\text{Bit Rate}} \) is not a whole number, it must be rounded off to the nearest integer.

The baud rate range is from 5 to 191.

The character size to be input is the least significant octal digit of the total number of data bits and stop bit(s) in a character and is either 1 or 2 depending on whether the character contains 1 or 2 stop bits.

For the IBM 2741 Terminal an "*" must be input for character size.

Error Conditions:

The message "ILLEGAL FORMAT" is output if

1. A baud rate <5 or >191 is input.
2. A character size other than 1, 2, or "*" is input.
3. A port number outside the range 0 through 31 is specified.

The message "NO CONF. DONE" is output if the port (configuration of a single port) or at least one port (configuration of more than one port) is logged on.

In each of the above error conditions no configuration will take place.

For each port to be configured the system first initiate service routine "CHS" in the I/O processor. "CHS" sets the character size (0 if an "*" was input!) into the receive and send parameter (?RPRM resp. ?SPRM) kept in the teletype table. If the character size = 0, the echo bit in ?RPRM is set to 0, otherwise it is set to 1. If the character size = 0, ?TYPE is set to 1 (indicating that an IBM 2741 terminal is connected to the port. Otherwise it is set to 0, indicating that an ASCII terminal is connected.
Then the system initiates service routine "SPE". This routine sets the new baud rate in ?SPRM and ?RPRM and sets the parity bit in ?SPRM to 1 ("EVEN" parity will be generated on output) if ?TYPE = 0. Otherwise the parity bit in ?SPRM is set to 0. Finally "SPE" retrieves ?SPRM and ?RPRM and output these parameters to the multiplexer board.
PORT
This routine is used to print out the baud rate and character size for which the port(s) is (are) configured.

The system processor obtains the information from the I/O processor.
If an IBM 2741 is connected to a port, an "*" will be printed for the character size.

Error Conditions:
The message "ILLEGAL FORMAT" will be given, if an illegal port number is specified. No information is given.

This command is available to the system operator and the system master.
If this command is given by a user other than the system master, the error message "privileged command" is output and no information will be given.

For each port the system uses service routines WSP and WCS to obtain the baud rate resp. character size for which the port is configured.
I/O PROCESSOR PROGRAM

The second computer in the 2000C high speed is used for all of the terminal input and output operations for the system. In addition, it takes care of all of the phones logic (answering and hanging up the telephones) and the timing for the ENTER STATEMENT.

2100A Asynchronous channel Multiplexor

To put the Multiplexor into operation, each port on the interface must be primed with two parameters. The parameters are necessary for transmission and reception of data to/from that port. One parameter is used for the send channel and one parameter for the receive channel of that port. Once primed, those parameters will remain in the channel's memory until the power goes down or a "master clear" is executed.

The parameters consist of 16 bits which have the following functions:

Send Channel Parameter

| Bits 0-7  | Indicates the rate at which the data bits will be transmitted. |
| Bits 8-10 | Indicates the least significant bits of the number of bits, including stop bits, in a character. |
| Bit 11    | Not used by the system. |
| Bit 12    | If set, ASCII parity will be generated. |
| Bit 13    | If set, interrupt on completion of transmission of data will be enabled. |
| Bit 14    | Must be set. |
| Bit 15    | Must be set. |

Receive Channel Parameter

| Bits 0-7  | Indicates the rate at which the data bits will be received. |
| Bits 8-10 | Indicates the least significant bits of the number of bits, including stop bits, in a character. |
| Bit 11    | Not used by the system. |
| Bit 12    | If set, all received data will be echo'd back to the terminal. |
| Bit 13    | If set, interrupt on reception of data will be enabled. |
| Bit 14    | Must be=0. |
| Bit 15    | Must be set. |
The Multiplexor consists of two boards, a "data" board and a "status" board. They must be located in "two consecutive I/O slots; the "data" board in the higher priority slot (lower numbered select code) and the "status" board in the lower priority slot (higher numbered select code). Output of data to a send channel (to be transmitted to the terminal) must be in the format.

Bits 0-10  Data*
Bit 11     Must be set, if a "synchronizing" character must be transmitted.**
           Otherwise, must be = 0.
Bits 12-13  Immaterial.
Bit 14     Must be set.
Bit 15     Must be = 0.

* The ASCII character must be contained in Bits 0-6. Bit 7 must be = 0 since even ASCII parity must be generated. Bits 8-10 must be = 1. The selectric character (6 bits/char.) must be contained in bits 0-5. Bit 6 must be set or reset according to the odd parity of Bits 0-5. Bit 7 must be = 0 and Bits 8-10 must be = 1.

** A "synchronizing" character is issued by the I/O processor to provide one character time delay. It is used to delay output until the terminal has completed a carriage return or line feed. The number of "synchronizing" characters is dependent on the terminal. The format of the "synchronizing" character is as follows:

Bits 0-6  Must be = 1.
Bit 7    Must be = 1 if terminal is a selectric.
        Must be = 0 for all other terminals.
Bits 8-10 Must be = 1.
Bit 11   Must be = 1.
Bits 12-13 Immaterial.
Bit 14   Must be = 1.
Bit 15   Must be = 0.

The "synchronizing" character is a non-printable character. Besides, providing a means of time delay, it is used to synchronize the terminal at the beginning of every transmission.
Input of data is done with a "LIA" or "LIB" to the "data" board. Its format is:

**Bits 0-6**  Data bits.

**Bits 7-9**  Immaterial.

**Bits 10-14**  Number of the channel on which the data was received.

**Bit 15**  Not used by the system.

An "LIA" or "LIB" to the "status" board gives information in the following format:

**Bit 0**  If=1, the interrupt came from the completion of a character transmission (send channel).

If=0, the interrupt came from the completion of a character reception (receive channel).

**Bit 1**  Not used by the system.

**Bit 2**  If=1, a "break" signal was received.

**Bit 3**  Not used by the system.

**Bits 4-9**  Immaterial.

**Bits 10-14**  Number of the channel on which an interrupt occurred.

**Bit 15**  "Seeking" bit. This bit indicates that a "seek" operation is taking place in the circulating memory of the interface. If=1, no data or parameters should be output to the interface.

Method of outputting the "send" and "receive" parameters to the interface:

1. "LIA upper select code"
2. Check seeking bit. If=1, the previous operation was not yet completed. Go back to 1. If=0, proceed to 3.
3. "OTA lower select code" (the parameter is assumed to be in the "A" register).
4. "OTB upper select code" (the channel number is assumed to be in bits 10-14 of the "B" register).
5. "STC lower select code".

Output of data is done in the same way.
MULTIPLEXER DRIVER

The multiplexer driver is used by both multiplexer boards. The driver is divided into five sections:

I. Initialization - One routine for each board
II. Receive channel processing
III. Send channel processing
IV. Abort processing
V. Multiplexer end of processing

I. Initialization

The initialization section has two interrupt entry points, MPXIO for the first board, and MPYIO for the second board. If entry to the driver is made at MPXIO, the registers are saved and then both MUX channels are read and saved. YFLAG is now checked to see if the lower priority board is currently using the driver. If the driver is busy, XFLAG is set, the registers are restored, and the program is exited. If the driver is not busy, no flag is set as the lower priority board cannot interrupt. A check of the multiplexer status determines which processing section (input, output, or abort) is needed to service the interrupt.

If entry is made at MPYIO, the registers are saved; the MUX channels are read and saved; and YFLAG is set. The multiplexer status determines the processing section.

II. Receive Channel Processing

The multiplexer supplies whole characters, each of which are examined on reception and echoed back to the terminal. If the user's terminal is an IBM Selectric (?TYPE ≠ 0), the character is first translated into ASCII. Certain characters (rubouts, feed frams, line feeds, and X-OFF) are ignored. 'Control X' signals that the current line is to be deleted. If the character is a ' ', the buffer pointer is backed up one position. If the user has the line printer as his output device: 'Control Q' causes suspension of output to the line printer; 'Control W' results in resumption of output to the line printer.
II. Receive Channel Processing, Continued

All other characters are appended to the user's buffer.

Upon reception of a carriage return, the system processor is notified that the user has entered a complete line and further character input is blocked. If the line was entered in response to an ENTER statement, the user's response time is also sent to the system.

III. Send Channel Processing

If there are characters left in the user's buffer, a test is made to see if there is line feed or carriage return delay pending. If so, a synchronizing character is output and the delay counter is bumped. If not, the user's next character is plucked from his buffer, translated to IBM code if warranted, and sent to his port. If the character was a line feed or a carriage return, the appropriate delay is set up. If exactly ten characters remain in the user's buffer and if his status is output wait, the system is notified that his buffer is almost empty.

If no characters remain in the user's buffer: He is placed in an idle mode if his program is still running; or he is placed in input mode.

IV. Abort Processing

Unless the aborted occurred on the receive channel with the user in output mode, it is ignored. For a valid abort, the abort request is sent to the system and the user's buffer pointers are reset to the beginning of his buffer.

V. Multiplexer End of Processing

Four interrupt combinations can occur. This logic determines which flags to clear, which multiplexer board to enable, and where to transfer program control.
LINE PRINTER DRIVER

This driver is used for the 2767A, 2778A, and 2610A line printers. Normal entry is from the idle loop and once entered, the driver replaces the idle loop until output is completed. The flag, LPTYP, indicates which line printer is on the system:

\[ \begin{align*}
2767A &= -1 \\
2778A &= 0 \\
2610A &= 1
\end{align*} \]

Characters are obtained from the user's buffer, and then tested before being sent to the line printer. If the character is a carriage return or a line feed, a print control character is output. If the character is an X-OFF and the next character is an X-OFF, line printer output is temporarily suspended. In addition, control characters and rubouts are ignored and lower case characters are converted to upper case.

If the line printer goes out of READY status, the user's buffer pointers are saved and new pointers are set to an error message buffer. Output to the user's teletype is then initialized and when the transmission has completed, the buffer pointers are reset and the driver waits for READY status.
2100 DATA SET CONTROL INTERFACE

The data set control board is used in the "SCAN" mode so that an interrupt will only occur if a change in either of the two signals ("carrier" and "Data set ready") has been detected. To prime the board for an interrupt is is necessary to output a parameter with the following format:

Bit 0  "Data set ready" bit. If = 0, an interrupt will occur when "Data set ready" comes up. If = 1, an interrupt will occur when "Data set ready" drops.

Bit 1  "Carrier detect" bit. If = 0, an interrupt will occur when "carrier" comes up. If = 1, an interrupt will occur when "carrier" drops.

Bit 2  Enable bit for comparison Logic. If = 1 and if the comparison Logic detects a change in "Data set ready", the flag will be set and scanning is stopped.

Bit 3  Enable bit for comparison Logic. Same as Bit 2, but applies to "carrier detect".

Bit 4  = 1 for "Data terminal ready" on.
       = 0 for "Data terminal ready" off.

Bit 5  Must be = 1.

Bit 6  Enable bit for "Data terminal ready". If = 1 and Bit 4 is = 1, "Data terminal ready" will be transferred to the interface.

Bit 7  Must be = 1.

Bits 8-9  Immaterial.

Bits 10-13  Channel Number.

Bit 14  If = 1, Bits 0-3 will be transferred to the interface.

Bit 15  Must be = 1 for operation in "scan" mode.
On an interrupt because of a change of "Data set ready" or "carrier" the obtained status has the format:

Bit 0
If = 0, "Data set ready" has dropped.
If = 1, "Data set ready" has come up.

Bit 1
If = 0, "Carrier" has dropped.
If = 1, "Carrier" has come up.

Bit 2
Has the same value as Bit 2 in the parameter, output to the interface.

Bit 3
Has the same value as bit 3 in the parameter, output to the interface.

Bits 4-7
= 0.

Bits 8-9
Not used by the system.

Bits 10-13
Number of the channel on which the interrupt occurred.

Bits 14-15
= 1.

After examining the status and taking the steps necessary to connect, disconnect, set up log-on timing, etc., the interface has to be primed for the next interrupt, based on new conditions of change in "Data set ready" and "carrier". This can be simply accomplished by outputting the obtained status to the board.
DATA SET CONTROL BOARD DRIVER

The driver for this board is used in the "scan" mode so that an interrupt only occurs when a change in the signals "Data set ready" (=CC) and "Carrier detect" (=CF) is detected by the board. As soon as an interrupt occurs, the new status of the channel is compared with the previous status which was saved in ?PPRM teletype table entry. Depending on that comparison one of the following will be executed:

1. The phone is answered, "LTBT" bit in ?STAT is set and log on timing of 120 seconds (subject to change by the system operator with PHOnes command) is stored in ?PHON.

2. "LDBT" bit is set in ?STAT and dropout timing of 2 seconds is stored in ?PHON.

3. "LDBT" bit is reset in ?STAT to signal that connection was restored within the 2 seconds dropout timing.

On exit the new channel status is saved in ?PPRM and output to the board.

NOTE: The "LTBT" or "LDBT" bit will cause the time base generator routine to start timing using the value of ?PHON as a counter.
INITIALIZATION

I. When the I/O processor program is started at 'INI' (initiated from location 2), the following is done:

1. Do a "master clear".
2. Set "CKFLG" to 0 (flag to be used by the power fail routine).
3. Initialize all 32 teletype tables:
   A. Set ?TYPE = 0 (to terminal type #1).
   B. Set CR-DELAY and LF-DELAY for terminal type #1 in ?CDLY resp. ?LDLY
   C. Set ?RPRM to 110 baud, char. size = 2 and echo bit on.
   D. Set ?SPRM to 110 baud, char. size = 2 and "even" parity.
   E. Set "Data term. ready" on, "req. to send" on, "carrier detect" off and "data set ready" on.
4. Set phones timing to 120 seconds.
5. Set "NPORT" = 32
6. Initialize processor interconnect board.
7. Initialize power fail board.
8. Go to idle loop.

II. When the I/O processor is updated at the system update entry "INIF", the following is done:

1. Do a "master clear".
2. Get number of available ports and save it in "NNPRT".
3. If "NNPRT" is larger than "NPORT", initialize the ports with port numbers between NPORT and NNPRT (see I, Sub 3). for ports with port numbers above "NNPRT" set the enable bit (Bit #13) in ?RPRM teletype table entry to 0 and set "Data term ready" off in ?PPRM teletype table entry. If "NNPRT" is less than "NPORT", set the enable bit in "Data term. ready" off in ?PPRM for all ports with port numbers above "NNPRT". If "NNPRT" is equal to "NPORT", go to 4.
4. Set in all teletype tables:
   A. ?CCNT = 0
   B. ?BPNT = ?BGIN
   C. ?BSTR = ?BGIN
   D. ?BHED = ?BGIN
   E. "IDBT" bit (Bit #3) = 0 in ?STAT
   F. ?DCNT = 0
   G. ?SCNT = 0
5. Output $?RPRM & $?SPRM to the multiplexor board(s) and $?PPRM to the
data set control board(s) for each port up to the port with port
number = "NNPRT".
6. Set "CKFLG" = 1
7. Store contents of "NNPRT" into "NPORT". ("NPORT" indicates now the
current number of available ports).
8. Initialize the time base generator board for 100 MS time interval.
9. Initialize the multiplexor board(s) and the data set control board(s).
POWER FAIL AND RECOVERY

If the computer is running when a power failure occurs, the current machine and I/O status is saved. A flag is set to indicate that this status has actually been saved and the program halts.

If the computer is halted when a power failure occurs, the power down interrupt does not occur.

When power is restored, the flag is checked to determine whether or not the power down interrupt was processed. If not, the initialization section (entry point "INI") is called. If so, the program restarts using the saved status and all of the ports buffers and status are retained. The parameters on the multiplexor board(s) and data set control board(s) are re-instated as before power failure. When power is restored, the line printer will be disconnected from a user who had control over it at the time of power failure.
TIME BASE GENERATOR

The time base generator driver is entered every 100 MS. For every available port the driver will scan for "LTBT", "LDBT", "HUBT", "ENBT" and "PDBT" bit in the ?STAT teletype table entry. If one of these bits is set, the following actions will be taken:

A. "LTBT" bit set.
1. Update timing counter in ?PHON for log on timing (See write up of data set control board).
2. If counter becomes zero, reset "LTBT", "LDBT" and "HUBT" bits in ?STAT. Then set "PDBT" bit and output "Data term. ready" off to the appropriate data set control board forcing the phone connection to be broken.
3. Also the "IOBT" bit in ?STAT will be set, ?TYPE will be set to 1 if type #2 terminal is connected; CCNT, ?DCNT and ?SCNT will be set to zero.

B. "LDBT" bit set.
1. Update timing counter in ?PHON for line dropout timing (See write up of data set control board).
2. If counter becomes zero, tell system processor that user has hung up (communication code "UHU").
3. Reset "LDBT", "ENBT" and "ICBT" bits in ?STAT.
4. Execute A, Sub 3.

C. "HUBT" Bit set.
1. Check ?CCNT indicating the number of characters to be output. If ?CCNT not = 0, check if "LTBT" or "LDBT" bits are set. If ?CCNT = 0, execute A, Sub 2 and A, Sub 3.

D. "ENBT" Bit set.
1. Check ?CCNT. If ?CCNT not = 0, check if "LTBT", "DBT" or "HUBT" bit is set. If ?CCNT = 0, update enter timing counter in ?TIMO.
2. If counter becomes zero, remove "ENBT" bit and set "NIBT" bit in ?STAT. If type #2 terminal is connected, set Bits 8 through 12 in ?TYPE. This will force the selectric into receive mode. Fetch ?RPRM from teletype table, set echo bit (Bit #12) to zero and output it to the multiplexor board.
2. (Continued)
Tell system processor that user was timed out (communication code "ETO").

E. "PDBT" set.
1. Reset "PDBT Bit in ?STAT
2. Tell the data set control board to stop scanning.
3. Take status on the data set control board for the appropriate channel. Output to the data set control board the phones parameter with "Data term. ready" on, "Req. to send" on, "carrier detect" = 0 (if status indicates that it is = 1) and "Data set ready" = 0 (if status indicates that it is = 0) or = 1 (if status indicates that it is = 1).
TELETYPE TABLES

The teletype tables are located in base page and contain information about the system users. Each of the 32 users has one table containing the following entries:

- **?TNUM**: Port number in Bits 8-12
- **?CCNT**: Used by MPX for counting output characters. It equals -# of characters, including current one.
- **?BPNT**: On input - Points to the character location into which the next character will be deposited.
  On output- Points to the last character transmitted.
- **?BSTR**: On input - Points to the first character of the most recent buffer.
  On output- Points to the location into which the next character will be placed by the outcr routine.
- **?BHED**: On input - Points to the next character to be fetched.
- **?BSAV**: Saved buffer pickup pointer.
- **?BGIN**: Points to beginning of physical buffer.
- **?BEND**: Points to first character following physical buffer.
- **?STAT**: 

```
TPBT   EQU BIT0
TPNBT  EQU NBT0
STBT   EQU BIT1
STNBT  EQU NBT1
CXBTE  EQU BIT2
IOBTE  EQU BIT3
IONBT  EQU NBT3
LDBTE  EQU BIT4
LDNBTE EQU NBT4
LTBTE  EQU BIT5
LTNBTE EQU NBT5
ENBTE  EQU BIT6
ENNBT  EQU NBT6
RNBT   EQU BIT7
RNNBT  EQU NBT7
```

User is in tape mode
User was turned off
'Control X' was hit
User is in input mode
Line dropout occurred
Wait for log timing
Timing for <ENTER>
User is running

141
**TELETYPE TABLES, Continued**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PDBT</td>
<td>EQU BIT8 Phone disconnected</td>
</tr>
<tr>
<td>NIBT</td>
<td>EQU BIT9 No input allowed</td>
</tr>
<tr>
<td>NINBT</td>
<td>EQU NBT9</td>
</tr>
<tr>
<td>HUBT</td>
<td>EQU BIT1 Hang user up</td>
</tr>
<tr>
<td>XOBT</td>
<td>EQU BIT11 X-OFF was read from terminal</td>
</tr>
<tr>
<td>STYP2</td>
<td>EQU BIT12 *</td>
</tr>
<tr>
<td>STYP3</td>
<td>EQU BIT13 *</td>
</tr>
<tr>
<td>STYP4</td>
<td>EQU B1213 ** Teletype subtypes</td>
</tr>
<tr>
<td>STYP5</td>
<td>EQU BIT14 *</td>
</tr>
<tr>
<td>STYP6</td>
<td>EQU B1214 *</td>
</tr>
<tr>
<td>ICBT</td>
<td>EQU BIT15 Input configuration needed</td>
</tr>
<tr>
<td>ICNBT</td>
<td>EQU NBT15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>?ATIM</td>
<td>Contains allowed time for &lt;Enter Statement&gt; execution.</td>
</tr>
<tr>
<td>?TIMO</td>
<td>Timeout value for user executing &lt;Enter Statement&gt;.</td>
</tr>
<tr>
<td>?PHON</td>
<td>Used as time counter for phones logic.</td>
</tr>
<tr>
<td>?TYPE</td>
<td>Terminal Type:</td>
</tr>
<tr>
<td></td>
<td>ASCII =Ø</td>
</tr>
<tr>
<td></td>
<td>EBCD Bit Ø =1</td>
</tr>
<tr>
<td></td>
<td>Bit 15 = Ø</td>
</tr>
<tr>
<td></td>
<td>Call/36Ø Bit Ø = 1</td>
</tr>
</tbody>
</table>

**FOR EBCD & CALL/36Ø TERMINAL:**

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDBT</td>
<td>EQU BIT1 Code determined</td>
</tr>
<tr>
<td>UCBT</td>
<td>EQU BIT2 Upper case mode</td>
</tr>
<tr>
<td>UCNBT</td>
<td>EQU NBT2</td>
</tr>
<tr>
<td>CNBT</td>
<td>EQU BIT3 &quot;Cent&quot; character</td>
</tr>
<tr>
<td>CNYBT</td>
<td>EQU NBT3</td>
</tr>
<tr>
<td>CCBT</td>
<td>EQU BIT4 &quot;CentC&quot; character</td>
</tr>
<tr>
<td>CRBT</td>
<td>EQU BIT5 &quot;CR&quot; Bit (Output only!)</td>
</tr>
<tr>
<td>CRNBT</td>
<td>EQU NBT5</td>
</tr>
<tr>
<td>XBIT</td>
<td>EQU BIT6 &quot;Control X&quot; was input</td>
</tr>
<tr>
<td>XNBIT</td>
<td>EQU NBT6</td>
</tr>
</tbody>
</table>
CBBT EQU BIT7  "Circle C" was sent
CBNBT EQU NBT7

BIT8  Circle D *
BIT9  SYNC    * Transmit
BIT10 Space   ** Interrupt
BIT11 Space   * Bits
BIT12 Space   *

?CDLY  Carriage return delay (negative).
?LDLY  Line feed delay (Negative).
?DCNT  CR and LF delay counter.
?SCNT  Character counter used for determining carriage return delays.
?RPRM  Receive channel parameters.
?SPRM  Send channel parameters.
?PPRM  Phone parameter.

Associated with each item in these tables is a symbol which is equated to the corresponding number of the item. For example:

?TNUM  EQU 0
?CCNT  EQU 1
.
.
.
?PPRM  EQU 23

These symbols are primarily used for adjusting pointers to the table. For example, if the B register contains a pointer to the STAT entry of some user, the instruction ADB .+ ?PHON-?STAT will point B to his PHON entry.

. is a symbol in base page at the 0 entry of a table of constants from -20 to +20. A word containing the value N, where -20 < N < 20, can be referenced by .+N.
SELECTRIC CONVERSION ROUTINES

There are two conversion routines in the I/O processor, both of which are entered from the multiplexor driver. The input conversion routine handles Call/360 or EBCD-to-ASCII conversion. The output conversion routine handles ASCII-TO-Call/360 or EBCD conversion.

The conversion routines use a set of bits which are stored in ?TYPE (See teletype tables).

Input Conversion

On entry a check of the "CDBT" bit is made. If it is not set, the code determination section is entered. In this section (if it is the 1st input character in the buffer) the input character is compared with the character "H" (in the "HELLO" command) in EBCD and Call/360 code. If it is the "H" in EBCD code, Bit #0 and Bit #15 are set to 0 resp. indicating that input came from an EBCD terminal. If it is not the "H" in EBCD code, Bit #0 and Bit #15 are set to 1 resp. 0 indicating that input came from a Call/360 terminal. Consequently if a user logs on with a 1st character other than an "H" from an EBCD terminal, his input will be treated as coming from a Call/360 terminal. The user's log on command will naturally not be recognized by the system so that the system will output "???", which will appear as "LLL" on the user's terminal. After exit out of the code determination section, the actual conversion routine is entered. If the user logs on correctly, the system will tell the I/O processor that user is logged on and in the "ULO" service routine the "CDBT" will be set.

If, on entry to the input conversion, the "CDBT" is set, the code determination section will be bypassed and the actual conversion routine will be entered.

Output Conversion

On entry a check is made if a "transmit interrupt" (See write up of Selectric terminal) has to be generated. If not, the actual conversion routine will be entered.
Both routines use the "CNBT" and "CCBT" bits if a character is input or must be output which requires a two or three character sequence (See manual).

If the selectric character "¢" is being input, the "CNBT" bit will be set. If the "¢" is immediately followed by a character as required by a two character sequence, the "CNBT" bit will be reset and the ASCII-equivalent inserted in the buffer. If not, the "CNBT" will be reset but nothing will be inserted in the buffer. If the "¢" on input is immediately followed by a "C", the "CCBT" bit will be set. (The "CNBT" bit was already set when "¢" was input). If the sequence "¢C" is followed by a character out of the Range A through Z, the "CNBT" & "CCBT" bits are cleared and the equivalent ASCII control character inserted in the buffer.

On output only the two character sequence is involved. If the "¢" is output, the "CNBT" will be set. If the "upper case code" (next character is in upper case and terminal is in lower case mode) or the "lower case code" (next character is in lower case and terminal is in upper case mode) is output, the "CCBT" bit is set. After output of the two character sequence, both bits are cleared.

The "UCBT" bit is set if an "upper case code" is input from the terminal or output to the terminal. If the "UCBT" bit is not set either a "lower case code" was input or output.

The "CRBT" bit is only used on output. The purpose of this bit is to prevent a line feed to be sent to the terminal if preceded by a carriage return. The reason for this being that the selectric is a "new line code" terminal. It will perform a carriage return + line feed on receipt of a carriage return.

The "XBIT" bit is used when the three character sequence "¢CX" (="X" in ASCII) is received from the terminal. The purpose of this bit is to let the I/O processor output "¢/" (="/" in ASCII) if the "¢CX" is immediately followed by a carriage return, the "XBIT" is also cleared but no action will be taken by the I/O processor.

The "CBBT" bit is set on output of the "Circle C" code to the terminal.

Bits #8 through #12 are used in the "transmit interrupt" operation.
CONVERSION TABLES FOR THE IBM 2741 TERMINALS TRANSMITTING EBCD & CALL/360 CODES

There are two conversion tables; one table for the EBCD-ASCII (and ASCII-EBCD) conversion and one table for the Call/360 - ASCII (and ASCII-Call/360) conversion. Each table consists of 177 locations, starting at a location pointed to by "CTBPI" (for Call/360-ASCII Conversion) resp. "CTBP2" (for EBCD-ASCII conversion). The organization of each table is as follows:

The upper part of each location is used for conversion on input, the lower part for conversion on output. The upper part of each location contains the ASCII-equivalent of the input character and the lower part the EBCD or Call/360 equivalent of the output character.

Method of Fetching the Character Equivalent

A. INPUT

Get the pointer to the appropriate conversion table and add to it the octal code of the input character (if the input character is in upper case, set Bit #6 of the sum just acquired). The sum is the address of the location in the conversion table where the character equivalent (ASCII) is stored. Fetch the contents of this cell, mask off the lower part; the remainder is the ASCII-equivalent of the input character.

B. OUTPUT

Get the pointer to the appropriate conversion table and add to it the octal code of the output character (ASCII). This sum is the address of the location in the conversion table where the character equivalent (EBCD or Call/360) is stored. Fetch the contents of this cell and mask off the upper part. The remainder is the Call/360 of EBCD equivalent of the output character.

If Bit #7 of the Call/360 or EBCD equivalent is set, that character is in upper case. Bit #6 is the parity bit (odd parity).

NOTE: The way to determine which conversion table to use, is to examine the ?TYPE entry of the teletype table. Bit #0=1 and Bit #15=0 for a Call/360 terminal. Bit #0=0 and Bit #15=1 for an EBCD terminal.
HARDWARE CONFIGURATION

I/O PROCESSOR

10  PROCESSOR INTERCONNECT
11  PROCESSOR INTERCONNECT
12  TIME BASE GENERATOR
13  1ST MULTIPLEXOR
14  1ST MULTIPLEXOR
15  DATA SET CONTROL FOR 1ST MULTIPLEXOR

16  LINE PRINTER (optional)

UP TO 16 TERMINALS

MORE THAN 16 TERMINALS

16  2ND MULTIPLEXOR
17  2ND MULTIPLEXOR
20  DATA SET CONTROL FOR 2ND MULTIPLEXOR
21  LINE PRINTER (optional)
NOTE: CABLE IS NOT SHOWN FOR CH2 - C1 CHANNEL. IT IS IDENTICAL TO THAT FOR C2 - CH1.
In the Idle State, the PI cards are set up as follows:

**C1(CH1)**
- CONTROL & ENCODE: SET
- FLAG & IRQ : CLEAR

**C2(CH2)**
- CONTROL & ENCODE: CLEAR
- FLAG : SET
- IRQ : CLEAR

A data transmission operation occurs thusly:

1) Sending machine waits for flag to be set on C2(CH2) indicating that the previous transmission has been processed.

2) Sending machine places data word in output register of C2(CH2) thereby placing it on the input register of CH1(C1). (OTA/B C2(CH2))

3) Sending machine issues STC, CLF to C2(CH2) making the ENCODE LINE go high, setting FLAG ON (CH1(C1)), clearing ENCODE on CH1(C1), and strobing data word into CH1(C1).

4) Sending machine issues CLC to C2(CH2) to prevent an interrupt from that card. The sending machine is now free to return to other tasks.

5) In the receiving machine, T5 will set the IRQ on CH1(C1). If the interrupt system is enabled and the priority line is high, the IRQ will cause an
Interrupt to a service routine.

6) The service routine does an LIA/B from CH1(C1) and decodes the 16 Bit data word.

7) If a response is called for, the receiving machine can load the output register with a data word (OTA/B CH1(C1)).

8) When the receiving machine has completed its processing, it issues an STC,C(CL1) to CH1(C1) which restores the cards to the idle state.

The following is the resultant statuses of the two computers after a command has been sent, received, and acknowledged:

a) The flag is set on C2(CH2) of the sending computer indicating that another transfer is now allowed. This occurred when the receiving computer issued an STC,C to CH1(C1) after it had decoded and executed the command. The STC,C is the acknowledgement to the SEND computer that the RECEIVE computer did receive the transmission.

b) The control on C2(CH2) is cleared by the CLC to C2(CH2). This was done to inhibit the interrupt that normally would occur after the SEND computer outputted the command.

c) The control is set and the flag is cleared on CH1(C1) (from the STC, C acknowledgement) indicating readiness to receive another transmission.
TWO PROCESSOR POWER FAIL CHARACTERISTICS

The two processors in the 2000B system have independent power supplies and consequently, power failure interrupts in either machine may occur at different times.

A problem arises if one computer is powered down, and the other machine attempts to send a transmission. Data will be lost as well as possible subsequent data transmissions. This is apparently caused by stray encode and data levels while power is coming up.

The internal consequences of a lost data transmission are these:

1. A line (Syntax, command, or input) being processed will be garbled.
2. Output characters will be lost. This problem will be hidden by the fact that the current output character is garbled (mux quits sending during character).
3. Under rare circumstances, such communications as echo-on, echo-off, phones-xx will be lost.
4. Terminals on which a carriage return has come in may never have that line processed by the 2116. The terminal will not accept input and the "Break Key" must be used to re-establish communications when power is returned.
5. The 2116 may loose the signal that indicates that the buffer for this user is almost empty. The terminal will stop typing and the program will remain in I/O suspend. The "Break Key" must be used to re-establish communications when power is returned.

6. The 2116 may loose the signal that indicates that the buffer for this user is full. The circular nature of the buffer will cause characters to be typed out of order. The probability of this error is almost zero.

7. If several users are typing on the 2114 and the 2116 is not running, all multiplexor activity may cease (2114 waiting for transmission to be acknowledged). This leads to the classic symptoms, i.e., no response to any struck key (even break), and termination of all output operations, perhaps with a space on the line (teletype chattering).

If the primary power source fails, the two machines will go down within milliseconds of one another and it is not so likely that any transmission will be in progress thereby being lost. If however, an individual processor's power is lost via a depression of the processor's power switch or a malfunction in its power supply one of the above symptoms is sure to occur if there is significant activity on the system.
If the 2116 is powered down first, the 4th, 5th, and 7th cases listed are probable. If the 2114 is powered down first, the 2nd case listed is probable. In other words, turn off the 2114 first if the system must be powered down. The two machines should not be turned off together.

If it is necessary to power down the system, and a common power switch does not exist, it is necessary to power down the 2114 prior to the 2116. Restart procedure dictates that the 2114 is powered up last.
## Core Allocation in I/O Processor Program

<table>
<thead>
<tr>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>Interrupt locations, variables, &amp; constants</td>
</tr>
<tr>
<td>0350</td>
<td>Teletype Tables</td>
</tr>
<tr>
<td>1550</td>
<td>System Processor Drivers</td>
</tr>
<tr>
<td>1700</td>
<td>Multiplexor driver routines</td>
</tr>
<tr>
<td>3000</td>
<td>Line Printer Driver Routines</td>
</tr>
<tr>
<td>3350</td>
<td>System Processor Service Routines</td>
</tr>
<tr>
<td>4600</td>
<td>Phones Logic</td>
</tr>
<tr>
<td>5000</td>
<td>Selectric Conversion Routines</td>
</tr>
<tr>
<td>6200</td>
<td>Time Base Generator Routines</td>
</tr>
<tr>
<td>6500</td>
<td>Initialization</td>
</tr>
<tr>
<td>7000</td>
<td>Power Fail</td>
</tr>
<tr>
<td>7700</td>
<td>Teletype Buffers</td>
</tr>
<tr>
<td>17700</td>
<td>250 Char. Each</td>
</tr>
<tr>
<td>17777</td>
<td>BBL</td>
</tr>
</tbody>
</table>
Initialization Section

MPXIO

SAVE: REGISTER MX STATUS

YFLAG = 0

N

Y

XFLAG = 1

RESTORE REGISTERS

MPXIO, I

MUX

ABORT?

Y

ABORT

N

SEND CHANNEL

Y

MPXOP

NIBT = 0

N

MPXEP

Y

INITIALIZE POINTERS

A = CHARACTER

R1

Received Channel Processing
**MULTIPLEXER OUTPUT ROUTINE**

- **OUTPUT**
  - **MIBT** = 1
  - TURN OFF CHARACTER ECHO

- **TYPE = 0**
  - N
    - **TYPE**
      - BIT = 1
  - Y
    - **OUTPUT, I**

- **MUXOR**
  - **A** = DATA
  - **B** = UNIT ≠

- **1ST BOARD**
  - N
    - **BOARD 2**
      - OUTPUT:
        - DATA UNIT ≠
  - Y
    - **BOARD 1**
      - OUTPUT:
        - DATA UNIT ≠

- **MUXOR, I**
Abort Processing

abort

receive interrupt

Y N

NIST = 1

Y N

clear status bits

IOBAT → 1

XTNUM = LPTNY

Y N

clear line printer flags

XTNUM = LPERF

Y N

LPETF → 0

restore buffer pointers

output ack to system

MPXEP
Note: The Driver for the 2nd Data Set Control Board is identical. The entry point is "DS2#1". The storage locations are DSF#2, DSFP2, and DSST52.

Diagram of flowchart for data set control board.
DATA SET CONTROL BOARD (3 of 3)

C

CHANGED FROM 0 TO 1?

YES

NO

F

E

SET \texttt{LTBT}

SET \texttt{?PHN} TO VALUE OF "PHR"

A
CLEAR PDBT

STOP SCANNING MODE OF DATA SET CONTROL BOARD

GET STATUS OF DATA SET CONTROL BOARD

GET BASIC PHONES PARAMETER

"DATA TERMINAL READY" = 1
"DATA SET READY" = 0
"REQ. TO SEND" = 1
"CARRIER" = 0

"DSR" IN STATUS = 1?

SET BIT 0 IN BASIC PHONES PARAMETER

SET BIT 1 IN BASIC PHONES PARAMETER

STORE PHONES PARAMETER IN ?PPRM

OUTPUT PHONES PARAMETER TO DATA SET CONTROL BOARD

C

158B
TIME BASE GENERATOR (3 of 5)

CLEAR LTBT, LDBT # HUBT

SET PDBT

OUTPUT PHONES PARAMETER, NOT "DTR"=# AND "RTS"=#

Subroutine"TBGII"
TIME BASE GENERATOR (Y OF 5)
Subroutine "TBG11"

1. Set I0BT
2. Terminal type #2?
   - Yes
     - Set bit 6 and clear all other bits in type
   - No
     - Set ?CONT= 0
     - Set ?DENT= 0
     - Set ?SENT= 0

Exit
SELECTED INPUT CONVERSION (4 OF 9)

S

CMBT = Ø?

YES

SET "A" = CONV. CHAR.

NO

INPUT CHAR. = 478?

YES

SET "A" = 1418

NO

INPUT CHAR. = 568?

YES

SET "A" = 1338

NO

INPUT CHAR. = 518?

YES

SET "A" = 1358

NO

INPUT CHAR. = 578?

YES

SET "A" = 1348

NO

INPUT CHAR. = 148?

YES

SET "A" = 1778

NO

INPUT CHAR. = 1858?

YES

SET "A" = 338

NO

INPUT CHAR. = 1178?

YES

SET "A" = 1738

NO

T

CLEAR CNBT IN ?7PE

I
SELECTED INPUT CONVERSION (5 of 9)
SELECTRIC INPUT CONVERSION (6 OF 9)

M
CLEAR Bit 6 in CONV. CHAR.

SET "A" = CONV. CHAR.

I

L
SET XBit in ?TYPE

J
CLEAR XBit and SET Bit 8 in ?TYPE

SET ICBF in ?STAT

SET "A" = 3φ/8

I
SELECTIVE INPUT CONVERSION (1 of 9)

Diagram:

1. If CMBT = φ?
   - NO: Set CMBT in ?Type
   - YES: Go to step 2

2. If CNBT = φ?
   - NO: Clear CNBT in ?Type
   - YES: Go to step 3

3. If CMBT = φ?
   - NO: Go to step 4
   - YES: Go to step 5

4. Go to step 2

5. Go to step 1
SELECTRIC OUTPUT CONVERSION (1 OF 6)

[Diagram of flowchart with decision points and actions]

- If Bit 12 in TYPE set, clear Bit 12 in TYPE and output "SPACE" to MUX board.
- If Bit 11 in TYPE set, clear Bit 11 in TYPE.
- If Bit 10 in TYPE set, clear Bit 10 in TYPE.
- If Bit 9 in TYPE set, clear Bit 9 in TYPE and send "Synchronizing" char. to MUX board.
- If Bit 8 in TYPE must be set, clear Bit 8.

'S OPERATION IS CALLED TRANSMIT INTERRUPT.'
SELECTIVE OUTPUT CONVERSION (C or D)

B

[Flowchart diagram with decision and action boxes, showing the logic flow between B, D, H, and branch points involving CCBT and CNBT in ?TYPE set? with associated actions such as setting values and conditional branches]
SELECTRIC OUTPUT CONVERSION (4 of 6)

1. Bit. 7 in converted char. set? (NO)
2. UCBT in ?TYPE set? (Yes)
3. Set UCBT in ?TYPE
4. Set "A" = 378
5. Set "B" = 0
7. Set Bit 7 of "A"
8. Exit

1. Bit. 7 in converted char. set? (YES)
2. UCBT in ?TYPE set? (YES)
3. CLEAR UCBT in ?TYPE
4. H
5. CLEAR CNBT & CCBT IN ?TYPE
6. Set "B" = 1
7. Set "A" = 348
8. Exit

160D
SELECTRIC OUTPUT CONVERSION (5 OF 6)

Diagram:

1. E
   - ORBT IN ?TYPE SET?
     - YES
       - CLEAR ORBT IN ?TYPE
     - NO
       - D
       - SET OUTPUT CHAR. TO "NULL"
SUBROUTINE "SLCNV"

ADD POINTER TO EBCDIC TABLE AND OUTPUT CHAR.

ADD POINTER TO CALL/360 TABLE AND OUTPUT CHAR.

BIT 0 OF TYPE SET?

YES

NO

GET WORD FROM TABLE WITH SUM AS ADDRESS

RESULT IS THE CONVERTED OUTPUT CHARACTER.

EXIT

EXIT
POWER FAIL/ AUTO RESTART

POWF

POWER UP?

NO

SAVE A, B AND E REGISTERS TEMPORARILY

SET "POWFF=-1"

FAILED DURING RESTART?

NO

SAVE A, B AND E REGISTERS + POWER FAIL RETURN ADDRESS

SET C00RS-PENDING BIT IN "POWFF" IF DEVICE FLAG OF A DEVICE WAS SET

NO

CENTRAL INTERRUPT FLAG SET?

YES

STORE "STF 0" IN LOCATION "POWND"

"CLE 4"

"HLT 4"

STORE "NOP" IN LOCATION "POWND"
SUPPLEMENTARY NOTES ON BASIC

I. SYNTAX

The general process of analyzing an input to the language processor is displayed in the section on flow charts. The annotations in the listing explain the actions of the subroutines, while the core map and section on internal representation describe the objects/structures being created or manipulated. The BASIC syntax, in conjunction with the listing, explains the method of identification and recognition of legitimate BASIC statements from the input string.

II. Phase 2
A. Compilation

The preliminary section of CMPLE prepares for execution of the program following a successful compilation. Null programs require no processing. If a sequence number follows the RUN (e.g., RUN - 220) the interpreter's program counter is set to the first statement whose sequence number equals or exceeds the reference, otherwise it is set to the first statement of the user program. If the common area has not been allocated, ALCOM is called to compute the space needed and move the program accordingly. If the program is already compiled (SYMTB=SPTR≠0) PBPTR is set back to the first word following the format stack (FCORE) and phase 2 simply reinitializes all of the variables to undefined. If the program is semi-compiled (SPTR=0, SYMTB≠0) we may skip building the symbol table. Otherwise FILTB is set to 0 so PRNST will not terminate compilation by mistaking it for decompilation.

The symbol table is then built as explained in the listing (Refer to the flow chart for general logic flow and to BASIC Variable Storage Allocation for a visual example). Also, at this time statement number references are replaced by absolute addresses. This is facilitated by dividing the program into 32 parts and building an 64 word table in ERSEC containing the first statement number and address of each part. During compilation SPTR points to the program word being processed. Pointers to <FILES statements> are stored in
FLSTS and a count of them is kept in FILCT. An error in compilation will cause a call to DCMPL to restore the source form of the program followed by a call to the error routine. If after a successful compilation at least one <FILES statement> has been found, BASIC calls the system, which analyzes the <FILES statements> and builds the file table, filling all but the fifth, sixth, ninth, tenth, eleventh and fifteenth words of each entry.

The symbol routine has two entry points: SSYMT is used for functions and simple variables and ASYMT is used for array and string variables. Because the dimensionality of an array variable may not be known locally (e.g., MAT A=B) some symbols may have two entries. If this is the case, the "don't know" entry will always be further down in the table (i.e., have a higher core address) than its dimensioned counterpart.

B. Value

VALUE is responsible for detecting deficiencies in the symbol table, allocating storage for the values of symbols (i.e., building the value table and common area), and initializing the values of all variables except those in common. Only the last of these functions is performed if a program is already compiled when a RUN command is received. The process of building the value table is described in the listing. Note that for arrays in common, the declared dimensions in the <COM statement> are checked against those in the common area. If they match and the dynamic dimensions are consistent (i.e., less than or equal to the declared ones) then the values are left alone. Otherwise they are set to undefined and both sets of dimensions are set equal to those in the <COM statement>. For strings, the physical length is checked against the declared length and the logical length tested to be less than or equal to the physical length. If these tests fail the physical length is set to the declared length and the logical length is set to zero. Simple variables in common are left untouched.

Several errors may be encountered while building the value table. The occurrence of a null symbol (bit pattern of 0) in the symbol table means that an array symbol is used in the program, but never in such a way that its dimensionality can be determined. If the second word of a function entry is zero, no <DEF statement> for that function appears in the program. Arrays
of more than 5000 elements are not allowed. For all errors the program is decompiled before the call to the error routine.

C. Decompilation

Programs are decompiled when any error occurs during compilation, building of the file table, building of the value table, or when the program is to be modified or saved in the user library. Since in the first of these only a portion of the program is compiled, the pointer SPTR is used to determine how much to be decompiled (A fully compiled program always has SPTR pointing to the first word following the program). The program is moved so that SPROG=PBUFF (no common area). The process is explained in the listing.

D. The Routine PRNST

PRNST is used by both CMPLE and DCMPL to scan the program and skip over those portions not affected by compiling. PRNST assumes responsibility for recognizing extra <FILES statements> and <COM statements> that are out of order. If such an error condition is encountered, SPTR is set to point before the statement which caused the error (it hasn't been compiled). Then PRNST calls DCMPL, which calls PRNST. The statement causing the error is not seen this time, so PRNST and DCMPL can exit correctly.

III EXECUTION

A. Main Loop

Upon completion of the value assignment in phase 2, control transfers to XEC. FCORE saves a pointer to the first word following the format stack (used in repeated RUNS of a program). After printing the program name (unless the program was CHAINED to) XEC proceeds to initialize the file table. A buffer the size of a logical record is allocated for each file and pointers to the word following it are placed in words 9 and 10 of the file table. The first word of the disc address of the record in the buffer (word 5) is set to 100000 to indicate that no record is present. Word 11 is set to Ø, indicating that no end-of-record/end-of-file exit has been specified. Word 15 is set to 0 as a null protectmask. If the file is read-only a message to this effect is printed, following the program name, unless the program was CHAINED to.
Following the preparation of files the initial execution status is set. The initial execution stacks are claimed from free user space and pointers are set to the first constant of the first <DATA statement>, if such exists. The internal print position counter (CHRCT) is set to zero by outputting a carriage return. Phase 2 has already set the BASIC program pointer (PRGCT) to the first statement to be executed.

Execution of a statement simulates the execution of an instruction on a 'BASIC machine'. The sequence number of the statement referenced by PRGCT is saved for possible use by the error routine. PRGCT is advanced to reference the following statement. The type of the current statement is used to branch to the appropriate routine via a jump table. Individual statement routines return to the top of the loop.

B. Statement execution

<LET statement> execution consists simply of evaluating the formula, which is known to contain at least one assignment operator and to have type compatibility (numeric vs. string) by its acceptance by phase 1.

<IF statement> execution forks on the symbol following the IF. The construction 'IF END' causes the following: the file reference is evaluated and tested for existence as one of the program's requested files; if a legitimate reference, the statement reference following the THEN is placed in the end-of-file word of the file's table entry. If not 'IF END', the decision formula is evaluated and if true the statement reference replaces the value of the interpreter's program counter, PRGCT, via the GOTO mechanism.

<GOTO statement> execution consists of choosing a statement reference to replace the program counter. For simple GOTO's this is done trivially; for multi-branch GOTO's this is done by evaluating the index formula and choosing the statement reference in the corresponding list position. If the index value lies outside the list of statement references, the program counter remains unchanged.
<GOSUB statement> execution follows the pattern for the GOTO except that after choosing the new value for the program counter, the old value is saved on the return stack (stack overflow generating an error condition).

<FOR statement> execution opens an active program loop. The for-stack is searched for an entry with the same for-variable; if found, the entry is eliminated (i.e., the previous <FOR statement> with this for variable is closed). A new entry is set on top of the for-stack (extending the for-stack by six words if no entry was eliminated) and a pointer to the for-variable's value entry is put into word 1. Since the first formula in the FOR contains an assignment operator, the formula evaluator, FORMX, initializes the for-variable when it determines the initial value. A reference to the statement following the <FOR statement> is put into word 6 of the for-stack entry (the start-of-loop address). Words 2 and 3 save the result of evaluating the limit value formula. If a step size formula appears explicitly it is evaluated, otherwise 1.0 is taken as the step size. In either case the value of the step size is left in words 4 and 5 of the for-stack entry. The program counter is set to the statement following the associated <NEXT statement> and control transfers to the <NEXT statement> execution code to compare the initial and limit values (see flow chart).

<NEXT statement> execution decides whether to iterate a loop or close it. The for-stack is searched for an entry with the same for-variable. If none is found the statement is ignored and control passes to the following statement. If the entry is found, any entries above it (more recent entries) are eliminated; i.e., they are assumed to belong to nested loops which were not closed by exceeding their limit value but exited otherwise. The value of the for-variable is then incremented by the step size and the new value tested by subtracting the limit value and using the sign of the step size to determine whether a non-negative or non-positive result indicates 'success'. If the result is 'success', the program counter is loaded from word 6 or the for-stack entry (the reference to the statement following the <FOR STATEMENT>), If the result is not 'success', the for-stack entry is eliminated. At this point the program counter already points to the statement following the <NEXT statement> so exit is simply to the main execution loop.
<RETURN statement> execution merely loads the program counter from the top entry of the return stack. An error condition is generated if the return stack is empty.

<INPUT statement> execution assigns values to the input list for both INPUT and MAT INPUT. INITF = 0 and MCNT is meaningless when executing an <INPUT statement>; For MAT INPUT, INITF = -1 and MCNT holds the number (in 2's complement) of elements of the current array as yet unassigned values. IFCNT holds the ordinal number of the current item in the current record (Note that IFCNT is not cumulative over the entire execution of a statement requesting input unless the request is met entirely by one line from the teletype).

The general approach in execution is to determine the address and type of a variable in the input list and then attempt to satisfy it from the input record. When an error occurs in the above process, it is explained along with any necessary corrective action and the value assignment is attempted again, so that errors in the input record will not terminate program execution. For simple input if the next variable in the list is of numeric type its value table address is placed into SBPTR; for array input the base address of the array is put into SBPTR. After filling a simple variable the next variable from the list is taken and a new address generated; after filling an array element SBPTR has been advanced to the next element by the numeric input routine so no new address need be calculated. When MCNT rolls over to zero (an array has been filled) control exits to the MAT INPUT code, which may return with another array's base address in SBPTR and MCNT reset appropriately. If the input record is empty but the variable list is not yet exhausted a request for additional input is made (signified by '?' rather than the initial '?'). SERR is needed as a flag to indicate if under/overflow occurred while converting the latest numeric input, since the error message will have destroyed any additional information in the input record. When looking for a number, the input record is scanned for the first sign (+ or -), digit, or decimal point, which begins the number. Any other characters will be ignored except the ", which will generate a recoverable error.
String input requires fairly complicated analysis of the data transfer. If the string variable does not specify the transfer length (does not have a double subscript), then the next string in the input record is transferred in its entirety and the logical length of the variable set appropriately. If the next string does not fit, a message is printed and a new string value requested. If the string variable specifies the transfer length then exactly that much of the next string in the input record will be transferred, either truncated or extended by blanks as necessary to achieve the specified length. The 'next string' in the input record begins with the next non-blank character or, if it is a '', the following character, blanks included. The string ends with the first '' (which is not part of the string) encountered or with the carriage return (also not part of the string) if no '' appears.

Every data item in the input record must be followed by a comma or carriage return and a comma must be followed by another data item. Failure to observe the above will generate recoverable errors. INTMP holds the type of data being sought, INTMP = 0 for a number or INTMP ≠ 0 for a string, and is used by the error recovery code to prepare for the entry.

<ENTER statement> execution assigns a value to a string variable or a simple variable. If a '#' follows the ENTER, the user's port number (0-31) is assigned to the first variable. The <ENTER statement> is timed and the length of time it took to respond (in seconds) is assigned to another variable. The input analysis proceeds much like an input statement with one variable, with the notable exception that no error messages are printed. Instead, the response time variable is negated if an error occurs. If the user does not respond within the allotted time, the response time variable is set to -256. This is non-ambiguous since response times are between 1 and 255 seconds inclusive. Also, for string input leading blanks are non stripped off and quote marks are allowed as characters.

<READ statement> execution assigns values to variables in the list. FDATA is primed to obtain values from either a file of the <DATA statement>s, depending on the presence or lack of a file reference following the READ. A mismatch in type between the variable and the next data item, or a string too long to fit into its designated destination, will generate an error and terminate execution.
<PRINT statement> execution consists of identifying items in the print list and sending the appropriate media equivalent to the teletype or disc file. An initial file reference identifies the statement as a file write and turns off the end-of-line mode; its absence identifies and teletype write and turns on the end-of-line mode. A comma or semicolon turns off the end-of-line mode and generates enough blanks to advance to the next field of 15 characters, if a teletype write. A literal string is written as a string of characters, less quotes, and turns on the end-of-line mode if a teletype write. An END writes an end-of-file mark on the file; it cannot occur in a teletype write. Formulas in the print string are evaluated and the results examined. Formulas which are string variables evaluate to their contents, which is then treated as a literal string. If not a string variable but within a file write statement, the floating point value of the formula is written on the file in its two-word binary representation. If a teletype write, floating point values are converted to an ASCII character string of the decimal equivalent. TAB can only occur in a teletype write; the evaluation of the TAB itself produces the desired action, so the value returned is thrown away, along with a following comma if one exits. For a teletype write all formulas turn on the end-of-line mode. If the end-of-line mode is on after processing the last print item, a carriage return-line feed is printed (This can only occur in a teletype write).

Before writing a quantity BASIC insures that sufficient space is available to accommodate it. CHRCT keeps track of the current print position on the teletype line (0-71). If the character string sent to the teletype would require non-blank characters to be printed past position 71, a carriage return-line feed is output first and CHRCT set to 0. If an item sent to a file requires more words than remain in the current record, BASIC automatically advances to the next record if in serial mode or exits to the end-of-record code if in record mode.

<PRINT USING statement> execution sends formatted output to the teletype. TEMPl is set to point to the first operand, NCH is set to the number of characters in the format string for a partial string or 0 for a full string, B = > the first word of the format string and A = character of the string to start with. Then the formatter is called, and it takes care of
retrieving the operands, formatting their values, and outputting them. See description of formatter elsewhere.

<RESTORE statement> execution resets the pointers to the DATA block. Beginning at the statement specified, or at the first statement in the program if none is specified, the pointers are set to the first <DATA statement> found, or to the out-of-data condition if none is found.

<END statement> and <STOP statement> execution terminates the program run. Since each requested file has a one record buffer in core, the last record written on a file does not exist on the disc in its updated form. Thus END and STOP must force the buffer of each read/write file onto its proper disc block. Also, the last change date for each file must be updated if any records have been changed. Following this, the word DONE is sent to the teletype and control exits to the scheduler.

<CHAIN statement> execution consists of calling the CHAIN library routine to get the named program from the disc and start execution of it.

<ASSIGN statement> execution changes the file referred to by a specified file number. After interpreting the file name and file number and dumping the last record of the previous file, it calls the ASSIGN library routine to update the last change date for the previous file and put information for the new file in the file table. Control is transferred back to BASIC to set the return code in the variable specified for it and set the protectmask in the file table if one is specified.

<MAT statement> execution involves many disparate tasks. The forms of the <MAT statement> may be classified as array I/O, array assignment, array initialization, and the array functions TRN and INV. For conciseness in coding, all forms other than array I/O use some common program segments.

Array I/O prepares each array in the list in the same fashion. SBPTR is set to the dynamic dimensions of the array (base address -2) and the operator following the array identifier is picked up for examination. At this point
MAT PRINT USING calls the formatter just as PRINT USING does. The EVEXP routine in the formatter takes care of picking up the elements of the array one by one, in rows. MAT PRINT follows a separate path than MAT READ and MAT INPUT. The following operator is noted as spacing the elements (comma or end-of-statement) or packing them (semicolon). VCHK examines the array and generates an error if any of its elements have value 'undefined'. The dynamic row and column lengths are saved in 2's complement. If the MAT PRINT references a file, the array elements are written one by one in rows, each element in its two-word binary form. If the MAT PRINT references the teletype, rows are double spaced and the elements within a row are spaced or packed as noted above, each element in its ASCII decimal form. Both MAT READ and MAT INPUT redimension the array if the following operator is a left bracket (i.e., begins a matrix subscript). MCNT is set to the number of elements in the array, in 2's complement. MAT READ calls FDATA for element values while MAT INPUT transfers to the <INPUT STATEMENT> execution to obtain element values. MTØ acts as a flag for MAT INPUT, differentiating the first call for input from subsequent calls and saving the input character following the last element value used from the input record. After completing I/O on an array, a common section of code prepares the next array in the list or, if no more remain, terminates the statement execution. MAT INPUT returns to the input code to clean up there, MAT PRINT and MAT READ return directly to the main execution loop.

Array assignment consists of preparing the destination and source arrays and executing a loop which assigns the destination array elements one by one. The general procedure is to assign a jump to the element computation code to MOP, an exit address to MEXIT to use after completing the destination array, and a count of the elements to MCNT, in 2's complement. The code to compute an element returns to MLOP1, MLOP2, or MLOP3 depending on the number of arrays involved which require updating of the element address. Each operation checks the dimensions of the arrays involved to insure that the operation is well-defined; and all elements of the source matrices are checked to make sure none have value 'undefined'. Matrix multiplication does not use the element.
computation loop, instead it uses row and column counters to tell when it is done and computes destination array elements by innter products of the rows and columns of its source matrices.

Array initialization also uses the element computation loop. The initialization program first redimensions the destination array (if a matrix subscript is given) and then chooses the appropriate constant for the element values. IDN acts like ZER except that it insists that the destination array be 'square' and sets a special counter to choose 1.0 for the value of main diagonal elements.

TRN and INV are handled apart from the other matrix functions. For both of these, the elements of the source matrix are checked against the 'undefined value'. The source and destination matrices are then checked for transpositional compatibility. If TRN, then proceed to transfer the columns of the source matrix to the rows of the destination matrix.

INV uses the Gauss-Jordan algorithm with row pivoting. This procedure converts a copy of the source matrix into the identity matrix and converts an identity matrix into the inverse by applying the same set of operations to both. Since the source matrix is destroyed in the process, it is first copied into free user space and the copy treated thereafter as the source. A side effect of the copying produces the element of largest absolute value, which is used to compute a lower bound on the allowable magnitude of pivot elements. INV then calls IDN to set the destination matrix to an identity matrix, having the side effect of checking that the matrix is square.

Diagonalization of the source matrix and production of the inverse now proceeds on a row-by-row basis. The next unreduced column of the source is searched for the pivot element (the largest in magnitude). If necessary, rows are swapped to put the pivot element on the main diagonal (the corresponding rows of the destination matrix must also be swapped). If the pivot element is smaller in magnitude than the previously computed lower bound, the matrix is too nearly singular to invert and execution is terminated. Otherwise, the pivot rows of both matrices are divided through by the pivot element.
Now all other elements in the pivot column are eliminated by subtracting the appropriate multiple of the pivot row from each of the other rows. Advantage is taken of those pivot column elements which are already zero and of the fact that elements of the pivot row to the left of the pivot column have been set to zero by previous steps. After diagonalization of the source matrix and consequent creation of the inverse, the user space occupied by the source copy is released.

The other statement types are declarative in nature. Execution of them consists solely of skipping over to the statement following.
Upon entry, B contains the address of the format string and A contains an index describing which character of the string the formatter should begin with. (This is for substring expressions.) The variable NCH will be zero if the entire string from character (A) on is to be considered. Otherwise it will contain a character count.

The routine grabs off the carriage control character, if any, and saves it. It then searches for a delimiting character ('/', ',', ') or end of string) and processes the specification up to that delimiter. The characters of the specification are examined and stored 1 character/word on a stack. Replicators are converted to binary and negated. Flags are set to indicate string, integer, fixed or floating point specifications. Literal strings are outputted directly from the stack and absence of any flag being set indicates a blank specification. The stack is then processed from top to bottom and each character or binary replicator and character causes appropriate output.

Strings are handled in a straightforward manner and may be output only if the string flag is set. Numbers are converted from binary to decimal and are stored 1 decimal digit/word in a number holding buffer. For integer or fixed specifications, the numbers are stored with decimal exponent of 0 and output directly according to the specification. For a floating specification, the number is stored with a maximum of 7 digits to the left of the decimal point and the decimal exponent is set accordingly. The number is then output in a straightforward manner.

When the stack has been exhausted, the delimiter(s) are processed. If the end of the string has been reached, and there remain expressions to output, the string is reprocessed from the beginning. If the end has not been reached, the next delimiter is found and the specification is processed as above. If there were no more expressions to output the carriage control character is processed and execution terminates.
Grouped specifications are handled by saving pointers to the beginning of the group and upon notice of the end, returning and reprocessing the entire group.

Formatter Utility Routines:

MTLI expects an unpacked floating point number in MANT1, MANT2 and EXP and returns a number there which has been made greater than 1. EXPON holds the count of multiplications necessary to make the number greater than 1.

DTLI expects an unpacked floating point number in MANT1, MANT2 and EXP and returns a number there on return. The A register contains a count of the number of divides necessary to make the number less than 1.

ROUND expects an unpacked floating point number in MANT1, MANT2 or EXP and rounds the number in the number holding buffer.

OUTBC and OUTCL are self explanatory.

DSRCH searches the format string starting at character pointed to by DP for a delimiter. If one is found, it is returned in the A register, and DP points to its location in the string. If the end of the string is reached and no delimiter is found, DP points one character past the end of the string.

MCHAR expects the address of a character in the A register. It returns that character in the A register. If the 0-Bit is set, blanks are ignored and the first non-blank character after that address is returned. In this case, if a delimiter is reached, the address of this delimiter (i.e., DP) is returned in the A register.
EVEXP is responsible for extracting the next variable to be output by the formatter. FFLG determines whether this is a MAT PRINT USING or a PRINT USING statement. For matrices, the first time EVEXP is called it verifies that all array elements are valid and returns the first element. Subsequent calls to EVEXP return array elements one at a time on a row by row basis. Numerical values are returned in the A and B registers and strings are returned with a pointer in the A register and the number of characters in the B register. EVEXP also evaluates the functions TAB, LIN and SPA and then goes to the next operand.
NOTES ON THE ERROR ROUTINES

Errors are handled routine SERR, reached by a jump through the base page table beginning at SERRS. A JSB SERRS + i,1 signifies detection of error i. The alternative bases RERRS, FERRS and WERRS are conveniences to denote subsections of the table used for run-time errors, format errors and warning-only errors. After printing a format error message, the offending format string is also printed. The actions taken by SERR are explained in the listing; but notice that the 'BAD INPUT' error is singled out, its processing is completed by the input execution routine upon return from SERR.

Syntax errors detected while in tape mode are handled by accepting error pseudo-statements in place of the erroneous statements. Since these pseudo-statements will be replaced by any subsequently received statements with the same line number, provision is made in FNDPS, which returns the location of a statement when given its sequence number, to decrement the error counter (ERRCT) whenever the statement found is an error pseudo-statement (an error pseudo-statement will only be found by FNDPS when another statement with the same sequence number is ready to replace it). Over/underflows detected during number conversions in syntax mode cause warning messages to be issued only after accepting the statement, if it is otherwise correct. Since no printing can be done while in tape mode, the routine CHOUF suppresses setting of the flag and these potential errors are not reported when in tape mode.
### BASIC Core Maps

**SYNTAX (Phase 1)**

<table>
<thead>
<tr>
<th>Pointer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>USE+</td>
<td>Fixed, first word of user swap area.</td>
</tr>
<tr>
<td>PBUFF+</td>
<td>Fixed, first word of program space.</td>
</tr>
<tr>
<td>SPROG+</td>
<td>Fixed, first word of program.</td>
</tr>
<tr>
<td>SBUFA+</td>
<td>Variable, first word of statement being syntaxed.</td>
</tr>
<tr>
<td>PBPTR+</td>
<td>Variable, first word of program space not used previously accepted program statements.</td>
</tr>
<tr>
<td>SBPTR+</td>
<td>Variable, first word not used by statement being syntaxed.</td>
</tr>
<tr>
<td>SYNTQ+</td>
<td>Variable, first word of syntax stack.</td>
</tr>
<tr>
<td>SSTAK+</td>
<td>Variable, last word of syntax stack.</td>
</tr>
<tr>
<td>LWAUS+</td>
<td>Fixed, first word not in user swap area.</td>
</tr>
</tbody>
</table>

**System Base Page**

- Subroutine Entry Points and User Variables
- Previously entered Program Statements
- Current Statement
- Buffer (105 Words)
- Syntax Stack
- Available User Space
- BASIC and System

**User Swap Area (10240 Words)**
COMPILATION (Phase II)

**Compilation**

<table>
<thead>
<tr>
<th></th>
<th>System Base Page</th>
</tr>
</thead>
</table>
| USE+| Subroutine Entry:
      | Points & User   |
      | Variables       |
| PBUFF+| Common Area    |
| SPROG+| BASIC Program   |
| SPTR+| Symbol Table    |
|      | Available User Space |
| LWAUS+| BASIC and system |

**Value Storage Allocation**

<table>
<thead>
<tr>
<th></th>
<th>System Base Page</th>
</tr>
</thead>
</table>
| USE+| Subroutine Entry:
      | Points & User   |
      | Variables       |
| PBUFF+| Common Area    |
| SPROG+| BASIC Program   |
|      | Symbol Table    |
|      | File Table      |
|      | Value Table     |
|      | Available User Space |
| LWAUS+| BASIC and System |

SPROG - Variable, first word of program
SYMTB - Variable, first word of symbol table.
SPTR - Variable, word of program being processed.
FILTB - Variable, first word of file table.
VALTB - Variable, first word of symbol value table
(FILTB = VALTB if no <FILES statement> is in program)
PBPTR - Variable, first word available of user space.
SYMTB and SPTR are not changed after compilation.
FILTB and VALTB are not changed after allocating value storage.

179
### EXECUTION (Phase III)

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IFSS</td>
<td>Variable, first word of format stack.</td>
</tr>
<tr>
<td>FCORE</td>
<td>Variable, first word not used by Phase II</td>
</tr>
<tr>
<td>RTRNQ</td>
<td>Variable, bottom of return stack (first word preceding return stack)</td>
</tr>
<tr>
<td>RTNST</td>
<td>Variable, top of return stack</td>
</tr>
<tr>
<td>FORQ</td>
<td>Variable, bottom of for-stack (sixth word preceding for-stack)</td>
</tr>
<tr>
<td>FORST</td>
<td>Variable, top of for-stack (points to latest 6-word entry)</td>
</tr>
<tr>
<td>TMPST</td>
<td>Variable, top of temporary stack (points to latest 2-word entry)</td>
</tr>
<tr>
<td>OPTRQ</td>
<td>Variable, bottom of operator stack.</td>
</tr>
<tr>
<td>OPDST</td>
<td>Variable, top of operand stack.</td>
</tr>
<tr>
<td>PBPTR</td>
<td>Variable, top of operator stack.</td>
</tr>
</tbody>
</table>

FCORE, RTRNQ, and FORQ are not changed after initiating execution.

Entries on the operator and operand stack are one word each and interleave (i.e., alternate words belong to one stack). All stacks beyond the return stack grow and shrink as needed so long as user space is available.
BASIC Internal Representation

BASIC statements are represented internally by the sequence number followed by the length in words (including the sequence number and length words) followed by the statement body. The statement body is composed almost entirely of operator-operand pairs which occupy from one to three words each. Null operands and operators are used when necessary to maintain the operator-operand correspondence. The operator resides in bits 14-9 of a word; the operand uses bit 15, bits 8-0, and sometimes whole additional words immediately following.

'Structural' Operands

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td></td>
</tr>
<tr>
<td>Simple</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td></td>
</tr>
</tbody>
</table>

Bits 8-0 are generally divided into two fields as follows: a name field (bits 8-4) and a type field (bits 3-0). The name field holds a value between 1 and 32, corresponding to A-Z (for functions, corresponding to FNA through FNZ). A type of 0 identifies a string variable (e.g. 3,0 represents C$). Types 1 and 2 identify array variables of dimensionality one and two respectively (e.g. 4,2 represents D[@*]) while type 3 identifies an array variable whose dimensionality cannot be determined by its immediate context. Type 4 identifies a simple variable with no digit (e.g. 1,4 represents A) while types 5-16 identify simple variables whose names include the digit 0-9 respectively (e.g. 6,7 represents F2). Type 17 identifies a programmer-defined function (e.g. 32,17 represents FNZ).
'Constant' Operands

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4-16 8</td>
</tr>
<tr>
<td>1</td>
<td>17 8</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0-72 10</td>
<td></td>
</tr>
</tbody>
</table>

Parameter

A parameter (which can only appear inside a<br>Pre-defined Function <DEF statement>) differs from a simple variable<br>Formal Dimension/<br>Branch Address<br>List<br>set. The name of a pre-defined function may range, in the standard system, from 1 to 210 or 248 to 308 (TAB to COS or ZER to TRN). A flagged (bit 15 set) operand of 3 identifies either a formal dimension in a <DIM statement> or <COM statement> (value in following<br>word) or a branch address list (one or more statement sequence numbers in the following words<br>String Constant

A flagged operand of 0 indicates that the following two words hold a floating-point constant (all numerical constants within a program are so represented). The operator with internal code 1 is ", which signals the start of a string constant. The operand portion of the word has a value from 0 to 72 10, indicating the number of characters in the constant. The string follows, two characters per word, and the closing " is not explicitly represented internally.
The table below gives the internal representation of the BASIC operators. Those operators which manipulate the formula evaluation stack during execution have associated priorities. All numbers are in octal notation.

<table>
<thead>
<tr>
<th>Code</th>
<th>Priority</th>
<th>ASCII</th>
<th>Code</th>
<th>Priority</th>
<th>ASCII</th>
<th>Code</th>
<th>ASCII</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>(end-of-formula)</td>
<td>26</td>
<td>5</td>
<td>&lt;</td>
<td>54</td>
<td>FOR</td>
</tr>
<tr>
<td>1</td>
<td>&quot;</td>
<td></td>
<td>27</td>
<td>5</td>
<td>#</td>
<td>55</td>
<td>NEXT</td>
</tr>
<tr>
<td>2</td>
<td>,</td>
<td></td>
<td>30</td>
<td>5</td>
<td>=</td>
<td>56</td>
<td>GOSUB</td>
</tr>
<tr>
<td>3</td>
<td>;</td>
<td></td>
<td>31</td>
<td>(unused)</td>
<td></td>
<td>57</td>
<td>RETURN</td>
</tr>
<tr>
<td>4</td>
<td>$(file)</td>
<td></td>
<td>32</td>
<td>4</td>
<td>AND</td>
<td>58</td>
<td>END</td>
</tr>
<tr>
<td>5</td>
<td>(unused)</td>
<td></td>
<td>33</td>
<td>3</td>
<td>OR</td>
<td>61</td>
<td>STOP</td>
</tr>
<tr>
<td>6</td>
<td>(unused)</td>
<td></td>
<td>34</td>
<td>6</td>
<td>MIN</td>
<td>62</td>
<td>DATA</td>
</tr>
<tr>
<td>7</td>
<td>(unused)</td>
<td></td>
<td>35</td>
<td>6</td>
<td>MAX</td>
<td>63</td>
<td>INPUT</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>)</td>
<td>36</td>
<td>5</td>
<td>&lt;&gt;</td>
<td>64</td>
<td>READ</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>]</td>
<td>37</td>
<td>5</td>
<td>&gt;=</td>
<td>65</td>
<td>PRINT</td>
</tr>
<tr>
<td>12</td>
<td>13(1)</td>
<td>[</td>
<td>40</td>
<td>5</td>
<td>&lt;=</td>
<td>66</td>
<td>RESTORE</td>
</tr>
<tr>
<td>13</td>
<td>13(1)</td>
<td>(</td>
<td>41</td>
<td>11</td>
<td>NOT</td>
<td>67</td>
<td>MAT</td>
</tr>
<tr>
<td>14</td>
<td>11</td>
<td>+(unary)</td>
<td>42</td>
<td>11</td>
<td>ASSIGN</td>
<td>70</td>
<td>FILES</td>
</tr>
<tr>
<td>15</td>
<td>11</td>
<td>-(unary)</td>
<td>43</td>
<td>11</td>
<td>USING</td>
<td>71</td>
<td>CHAIN</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>, (subscript)</td>
<td>44</td>
<td></td>
<td>IMAGE</td>
<td>72</td>
<td>ENTER</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
<td>=(assignment)</td>
<td>45</td>
<td></td>
<td>COM</td>
<td>73</td>
<td>'IMPLIED' LET</td>
</tr>
<tr>
<td>20</td>
<td>7</td>
<td>+</td>
<td>46</td>
<td></td>
<td>LET</td>
<td>74</td>
<td>OF</td>
</tr>
<tr>
<td>21</td>
<td>7</td>
<td>-</td>
<td>47</td>
<td></td>
<td>DIM</td>
<td>75</td>
<td>THEN</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>*</td>
<td>50</td>
<td></td>
<td>DEF</td>
<td>76</td>
<td>TO</td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td>/</td>
<td>51</td>
<td></td>
<td>REM</td>
<td>77</td>
<td>STEP</td>
</tr>
<tr>
<td>24</td>
<td>12</td>
<td>+</td>
<td>52</td>
<td></td>
<td>GOTO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>&gt;</td>
<td>53</td>
<td></td>
<td>IF</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Some examples of BASIC statements in their internal form are given below. Note that actual function parameter formulas, <DEF statements> formulas, and subscript formulas appearing in <MAT statements> require end-of-formula operators to signal their end whereas most formulas end either with the first operator which does not manipulate the formula evaluation stack or with the end of the statement. Note also that constants are considered signed only within a <DATA statement>. ASCII numbers are decimal, internal numbers are octal in the presentation below.

10 LET W1 = Y = (B = C) + 3*A[1,J+K]

<table>
<thead>
<tr>
<th>sequence number</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
</tr>
</tbody>
</table>

0  | 46 | 27 | 6 | LET W1 | 24 |
0  | 17 | 31 | 4 | = Y    | 14 |
0  | 17 |     |   | =      |    |
0  | 13 | 24 |   | ( B    | 0  |
0  | 30 | 34 |   | = C    | 1  |
0  | 10 |     |   | )      | 5  |
1  | 24 |     |   | +      | 0  |
0  | 30 |     |   | 3.0    | 1  |
0  |     |     |   |        | 3  |
0  | 22 | 12 | 2 | *A     | 1  |
0  | 12 | 11 | 4 | [1     | 14 |
0  | 16 | 12 | 4 | ,J     | 0  |
0  | 20 | 13 | 4 | +K     | 0  |
0  | 0  |     |   | (end-of-formula) |
0  | 11 |     |   | ]      |
```
36 DEF FNC (X) = X + A0

50 REM ARK

50 GOTO A OF 10, 20, 30

60 DATA -1, "ABC"
```

```
106 MAT READ #K;A[I]
```

PROGRAM FRAGMENT

<table>
<thead>
<tr>
<th>DEF</th>
<th>FNC</th>
</tr>
</thead>
<tbody>
<tr>
<td>(</td>
<td>X</td>
</tr>
<tr>
<td>)</td>
<td>X</td>
</tr>
<tr>
<td>-</td>
<td>X</td>
</tr>
<tr>
<td>+</td>
<td>A</td>
</tr>
<tr>
<td>+</td>
<td>C</td>
</tr>
</tbody>
</table>

VALUE TABLE FRAGMENT

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00000</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00000</td>
<td>00000</td>
<td>00000</td>
</tr>
<tr>
<td>00000</td>
<td>00000</td>
<td>00000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000</td>
</tr>
</tbody>
</table>

value of simple variable declared dimensions dynamic dimensions
active elements inactive element physical length/logical length character string
The symbol table consists of two-word entries, one for each unique symbol occurring in the user's program. The first word of an entry is the internal representation of the symbol as previously described. The second word of the entry is a pointer to the value of the symbol. For a programmer-defined function the value is the defining formula in the <DEF statement>. The value of a simple variable is a two-word floating point number. The value pointer of an array is its base address (i.e. the address of its first element); when an array is dynamically resized to occupy less than its physically allocated storage, it occupies a contiguous block justified to the low core portion of its element space. Since array symbols may not have dimensionality locally defined (e.g. MAT A=B), array symbols may have a "don't know" entry in the symbol table in addition to the dimensioned entry. Both entries have the same value pointer. The declared and dynamic dimensions occupy the four words preceding the element space in the value table. The value of a string is also its base address. A string is a character array (packed two elements per word in contrast to the two words per element for numerical arrays). Its physical (declared) length and logical (dynamic) length occupy the word immediately preceding its value space.

The value table and common area are simply the concatenation of the values for the symbols in the program, excepting programmer-defined functions.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read-only bit</td>
<td></td>
</tr>
<tr>
<td>Dirty record bit</td>
<td></td>
</tr>
<tr>
<td>Dirty file bit</td>
<td></td>
</tr>
<tr>
<td>Number of records in file</td>
<td></td>
</tr>
<tr>
<td>Logical record size</td>
<td></td>
</tr>
<tr>
<td>Disc or drum address</td>
<td></td>
</tr>
<tr>
<td>of last logical record</td>
<td></td>
</tr>
<tr>
<td>Disc or drum address</td>
<td></td>
</tr>
<tr>
<td>of record in file buffer</td>
<td></td>
</tr>
<tr>
<td>File base disc</td>
<td></td>
</tr>
<tr>
<td>or drum address</td>
<td></td>
</tr>
<tr>
<td>EOF/EOR exit address</td>
<td></td>
</tr>
<tr>
<td>File name</td>
<td></td>
</tr>
<tr>
<td>Protect mask</td>
<td></td>
</tr>
</tbody>
</table>

**FILE BUFFER**

Length specified by second word in file table
FILE TABLE

The file table consists of one fifteen-word entry for each file or place-holder ("**") in the FILES statements. Bit 15 of the first word is set if another user had read/write access to the file when it was requested (except for Axxx users) or if the file is a library file not being accessed by its owner. Bit 15 of the second word is set when an item is stored in the buffer, so that only records which are changed will be written back to the disc or drum. Bit 14 of the second word is set when a record is written to disc or drum and is used during program termination as a basis for updating the last changed date word in the file's directory entry.

A logical record-sized buffer is associated with each file table entry, and is accessed through pointers in the entry. An intra-buffer pointer designates the next portion of the record to be written or read. A fixed pointer to the first word not in the buffer acts as a bound on the intra-buffer pointer.

FILE CONTENTS

There are 4 data types possible in a file. A string has bit 9=1 and the length in characters in the lowest 7 bits of the first word, followed by the string packed 2 characters per word. A two-word floating point number has the upper two bits of the first word different, except for a zero, which has both words zero. An end-of-file is a -1, and an end-of-record is a -2, in the first word.

Data written to or read from a file is first exclusively ORed with the fifteenth word of the file table entry. This has no effect, of course, unless that word is nonzero. It will be nonzero only if an ASSIGN statement has been used to specify the file, and the statement included a protect mask parameter. End-of-file marks, end-of-record marks, and the first word of strings are not masked. Floating point numbers are masked when they are written to or read from the file buffer. Strings are masked when the buffer is read from or written to the disc or drum.
Update Last Change Date Routine

Each file and program entry in the directory has a word containing the hour of the year when the entry was last changed. It is necessary to update this word for files when a program is terminated for any of the following reasons: normal termination, CHAINing to a new program, error termination, abort and when a SLEEP or HIBERNATE command is issued.

The DFCHK bit in the user's ?FLAG word in his TTY table is set to 1 if there were any files statements in the program. This determines whether the LCD routine will be called. When it is, each file table entry is examined. If bit 14 of word 2 = 1, the file has been written on, so the last change date must be updated. This bit is set by the WRBUF routine.

The only abnormality in calling LCD occurs following an abort. The user is taken off the queue and re-inserted with priority 0 to run a core resident routine called ABUCD which writes the user to the swap track, calls LCD, and returns to the scheduler to finish aborting the user.
BASIC Run-Time Stacks

Return Stack

RTRNQ+

return address

RTNST+

9 words

The return stack is of fixed length, holding from 0 to 9 one-word entries at any time. An entry is the absolute address of the statement following the GOSUB which placed the entry on the stack.

For-Stack Entry

pointer to value of for-variable

limit

value

step

size

two-word floating point numbers

The for-stack is of variable length, containing one six-word entry for each for-loop which is currently active. Since the limit value and step size are kept in the entry, they may not be changed within the for-loop. The value of the for-variable is the one kept in the value table, so this may be altered by statements within the for-loop.

Program Fragment

<FOR statement>
succeeding statement

191
All operands (checked words) are addresses (i.e., C represents a pointer to the value of the simple variable C). Bits 7-0 of an operator entry contain the operators identifying code (See 'Basic Operators' Table) while bits 15-8 contain the operator's priority. Note the alternate-word structure of the stacks. The temporary stack holds intermediate values during the formula evaluation.

192
BASIC Language Processor Tables

The two areas of core labelled SBJTB and USER contain the mechanism allowing different users to exercise different portions of the language processor without interference. The language processor makes its subroutine calls to the labels in the area beginning with USER. The word following a subroutine entry point is an indirect jump through the appropriate address in the area following SBJTB. When a user is displaced by the system, his registers are saved at USER and the area of core from USER to PBPTR,L inclusive is dumped onto his track of the disc. Thus, a complete record of the language processor's status with respect to him is preserved. The only thing particular to a user which remains when he is swapped out is his own teletype table.

The tables headed by PDFTB (which must be in base page), SYNTB, XECTB, and FOJT are jump tables. The method in the last three cases is to compute a decision number, add the base address of the table, and transfer through the entry thus designated. The pre-defined function table is used by the formula evaluator to enter the code for evaluating pre-defined functions.

The tables headed by QUOTE and MCBQS have several uses. Their entries are explained in the listing and their use will be explained in those routines which access them. The Error Jump Table (at SERRS) is explained along with the error routines.
When PENS encounters the end of statement condition it exits to make

END OF PROGRAM? P1

LAST STATEMENT END? N

ALL FOR'S MATCHED N

DCMPL DECOMPILE PROGRAM

ERROR EXIT

E

NLFLG E 0
FILTB = END OF SYMBOL TABLE

ARE WE COMPILED FOR SAVE? Y

RETURN TO SAVE ROUTINE

F

ANY FILES STATEMENTS?

FILES

FILES

ANY ERRORS?

EXIT TO VALUE+2

ERROR EXIT
VALUE (5 of 5)

VS

SET POINTER TO COMMON AREA IN SYMBOL TABLE

 Physical Row Dimension ≠ Declared Dimension

 N

 Y

 Physical Column Dimension ≠ Declared Dimension

 N

 Y

 Compute Size of Declared Array

 Too Large?

 Y

 N

 Compute Size as Specified by Dynamic Dimensions

 < Declared Size?

 Y

 N

 Update Common Pointer

 VI

 Set Physical and Dynamic Dimensions = Declared Dimensions

 Compute Number of Elements in Array

 Too Large?

 Y

 N

 DCMPL

 DECOMPILE PROGRAM

 ERROR EXIT

 Initialize Array to 'UNDEFINED'

 Update Common Pointer

 VI
A. Main Loop

1. EXEC
   - ABC/HR
   - CHECK FOR ALERTS

2. WAS PROGRAM CHAINED TO?
   - Y
   - CLEAR CHAIN FLAG BIT
   - PRINT PROGRAM NAME
   - CONTINUE
   - ENDFILE
   - ASIMP
   - 0

3. ANY MORE FILES?
   - N

4. FILE READ ONLY NOT CHAINED TO?
   - N
   - ALLOCATE BUFFER AND INITIALIZE FILE TABLE ENTRY
   - MOVE TO NEXT FILE

5. Y
   - OUTPUT 'READ-ONLY' MESSAGE

6. SET PROGRAM CHANNEL TO?
   - N
   - PRINT CR-LF-LF
   - SET STACK POINTERS AND CLEAN UP
   - SETDP
   - SET POINTER TO FIRST DATA

7. Y
   - SAVE CURRENT STATEMENT'S SEQUENCE #
   - ADVANCE PROGRAM COUNTER TO NEXT STATEMENT
   - FIND CURRENT STATEMENT'S TYPE AND ADVANCE TO APPROPRIATE ROUTINE
B. Selected Statement Types

GOTU

simple branch? YES

load branch address

GOSUB

YES  PUT program counter on return stack

evaluate branch index

yes within label list? YES

ERROR
2000C Loader

The 2000C TSB Loader is a separate program which runs on the system computer. It is explicitly loaded by the operator to perform the following functions:

1) Generate a new system from paper tapes.
2) Update an existing disc resident system using new paper tapes.
3) Reload a system from magnetic tape.
4) Restore the drum resident bootstrap routines.
5) Resuscitate a system that has crashed because of certain hardware or software failures.

The loader is implicitly loaded when the operator requests any of the following functions:

6) A normal load from disc of a slept system.
7) A selective load.
8) A selective dump.
9) A sleep or hibernate.

In addition, the loader contains the moving head disc driver for the system, which remains in core at all times.

The operation of the loader for each of these uses may be understood by using the following brief descriptions, the following flowcharts, and the listing.

1) The loader generates the system tables by asking the configuration option questions. Discs are checked for labels and the drum resident tables are written out. The first system record is read, and, using it, enough disc space is reserved for the system. The remainder of the system is read from paper tape and written to disc.
The system library routines are handled by the SYSLB routine, which puts them on the disc. The remainder of the system is written out afterwards. The user swap areas are initialized and the DATE-TIME sequence is entered.

2) Paper tape update uses only the paper tape load section of the load sequence. It does not change any disc area other than the system code.

3) Magnetic tape load is similar to paper tape load except that certain tables are read from tape rather than being initialized. Also, after the system has been written to disc, the mag tape loading section is entered to load the users library.

4) The loader may be entered at 14000B to restore the bootstrap loaders to the drum. The bootstrap sequence is entered after they have been written.

5) When the system tables are intact (both in core and on the drum) but the system cannot be slept normally, the loader may be loaded and started at 3000B to force the sleep procedure to occur.

6) The bootstrap procedure calls in the loader which copies system information to the drum, reads in the system, and enters the DATE-TIME sequence.
7) If the selective load option is selected while loading, the mag tape load section performs the load and continues the load process as if it were a reload from mag tape.

8) Selective dump uses the mag tape dump section of the loader to generate a tape containing specified user library entries.

9) The sleep process first copies all of the system tables and sanctified files from the drum and core to the disc. Then, if sleep or dump has been specified, the system and system tables are dumped in their entirety to mage tape and the mag tape dump section is entered to dump the user library. If sleep, only entries changed since the last hibernate are dumped. If hibernate, all entries are dumped.
A different version of the 2000C Loader exists for each different type of moving head disc supported. Except for the disc driver and several configuration options default values, these loaders are identical.

This section contains brief descriptions of minor routines in the loader and flowcharts for the more complicated ones.
TTY 35

The loader teletype driver is a non-interrupt driver which provides output and input capabilities on the console (usually an ASR 35). On output requests, printing is completed before control is returned to the caller. On input requests, control is not returned until the operator has typed a carriage return. Character backup (using the left arrow) and line cancellation (using the escape key) are handled internally to the driver.

DRUMP

The loader drum driver is a non-interrupt driver which provides output and input capabilities on drums. The driver decodes the specified drum address into the proper select code, track, and sector numbers using the 7TBL part of the equipment table. Any drum errors are retried by the driver up to 100 times. Continued failure causes a halt (if DISC8 is zero) or a skip return (if DISC8 is an RSS) with an error message printed.

DISCZ

This routine provides the necessary environment for the moving head disk driver and transfers input/output/seek requests to the actual driver.

Since the interrupt system must be enabled for the disk driver, this is done in DISCZ.

In conjunction with the GMQBD routine, DISCZ provides a buffer for disk driver generated error messages and prints any such errors that occur.
JDSE

This short routine sets up all interrupt locations associated with moving head disks to call the driver interrupt processor.

SELC

Enter with the first character in (A). Routine finds a two digit octal number in the range [SELCD,1, SELCD+1,1]. Errors are printed, exit to P+3. Otherwise return to P+4 with the integer in (B) and the next character in (A).

GTDDNO

This routine searches the input record for "-" followed by a disk number followed by a comma. If A = -1 when called, a check is made to ensure that the specified unit exists. Errors are printed, exit to P+1. Otherwise, return with number in (A) to P+2.

SETDS

The SETDS routine updates the drum table and trax tables given the logical drum number and the new select code.

STDIS

This routine updates the disk table and interrupt locations given the logical disk number and the new select code and unit number.

GTTRK

This routine searches the ADT for a full drum track. The starting location is given in (A) (0 sez full search). No find causes an error (insufficient drum space) and loading is terminated. The entry is removed from the ADT.
FAPD

This routine searches the ADT for a piece of drum space of specified size. No find causes no skip return. On find, the space is removed and the return is single skip.

FNZSC

This routine advances TEMP3 to the next entry in the disk EQT that contains a valid disk. No find sets pointer to first entry, no skip return. A find sets other pointers and gives a single skip return.

RDLBL and WTLBL

These routines calculate the address of a specified disk label and call DISCZ to read or write the label.

SDADT

This routine is called by FSDAD to remove space from the ADT in core.

CDATE

This routine is used to allocate disk space for the tables. 8K is allocated for each track of a table.

GNDAT

GNDAT is used when system space is being allocated. It writes out the current disk ADT and reads in another. If no other exists, exit is to the out of storage error and loading is terminated.
BUADT

BUADT uses the trax table to generate the ADT for all of the drum space.

RTADT

This routine is used by CLDT to return any unused quarter tracks to the in core ADT.

RCOP

RCOP asks "configuration options" and waits for an answer of yes or no. The variable COFLG is set to 0 for no and non-zero for yes. Return is single skip for yes.

RQINT

This routine packs a one or two digit decimal integer in a specified range. An invalid input prints "ILLEGAL INPUT" and returns to P+1. A simple CR returns to P+2. Otherwise, return to P+3 with the integer in (A).

CFFW

CFFW converts the first 4 words of a directory entry (ID and NAME) into printable ascii format in a specified buffer.

GMQBD

This routine provides a placeholder for the disk driver when it asks for an error message buffer. Either this routine or DISCZ will print out the message.
WDLTD

This routine asks the question: "DISK OPERATING SYSTEM PRESENT?" (unless no configuration options) and uses this bit of information to place the two drum bootstraps onto the drum. If a properly configures DOS exists, the bootstrap will not be affected; however, it is still the operator's responsibility to lock any other tracks used by DOS.

WDLTE

This routine writes the final disk bootstrap onto each of the moving head disks.

BSBSO

BSBSO is the standard RTE, DOS and TSB.bootstrap for drum sector zero. If switch 0 is up when it is run, it reads in sector 2, otherwise sector 1, and jumps to the first location of that sector.

BSLDR

BSLDR is the TSB bootstrap that resides on sector 1 (sector 2, also, if no DOS). It reads the final disk bootstrap from block 1 of the moving head disk in set code 17 unit 0. The code in this loader was originally meant to be read in at an origin of 1008. (BSORG EQU 1008). Later versions of the loader load it at 146008 but the special construction remains.
RESU

This section of code is used for the resuslitation process. It reads in the system segment table and the DAT and calls the sleep section.

EOTCH

This routine checks for an end of tape (during Mag Tape Sleep operations). It is called when end of tape is not allowable (before first file mark). An end of tape prints a tape too short message and halts. Pressing run restarts the dump.

DREN - ENSU

The DREN routine is used to insert a 12 word directory entry into the directory. If the proper directory track is full, ENSU is called to redistribute the dir. tracks. See the system documentation for supersave for a description of its operation.
To process a DRUM command,

LDB9

GETDN pick up the drum number

SECOD pick up the select code

is the drum number = 0? 

is the select code 14? 

print " Drum 0 most is 14 " 

SETOD set up the appropriate tables,

LDB8
LD2WI

configuration option?

no

LDRIY

RLUM

do lock and unlock stuff.

RNRT

get the number of ports.

get the number of disc tracks per disc.

get the number of ID tracks.

LDWIY

BUADT

build the drum ADT

LD2Y
FSDAD will not remove the space on this call.

FSDAD get length and call for space.

alternate allocation option?

have we checked all of the MAP's?

print test for null name?

update & write out the correct directory track & direct table.

more entries in directory?

selective load?
the directory track was updated at the beginning of this entities loading. It has remained in cache during that loading. Any load failure would cause the loader to use the non-updated version still on the disk.

RSDAD is called with the same parameters used earlier. This time, however, it actually removes the space.

read in the appropriate ADT and tell DIABLE it has no good buffer

write back the modified ADT.

every entry left in the directory?

any entries left in the directory?

any SD's in the selective load table?
an unexpected end of file was discovered.

print "unexpected EOF.
exit: remove lost"

tell dirp not
to use the
old track
currently in
core & updated.

a tape error has occurred
on a user's
entry.

print
"tape error,
skip name list"

tell dirp not
to use the
updated directory
track now left in
core.

M7RD
read a
tape record

File
mark

other

SPDC

SPFM

SPFM

SPDC
1.1

SSFBG

FA FD
get down
space for
this entry

read in the
entry, tell drive
about tracked field

write the
entry onto
the drum

update the
inverse drum
setting

SSFBG

SSFBQ

SSFBT

SSFBQ

SSFBQ

SSFBA

SSFBS

Clear the
drum address
word

print "directory
descended"

write back
updated dir.
track

OKAY

SSFBQ

SSFBQ

SSFBQ

SSFBQ

write out the
new ADT
HLS

SPFS

RLDC
ask for
load or dump
commands

read in the
pieces of the
system overlay
by buffers

initialize

read next
system lib.
program from
the disk

SYSLB
writes sys
lib program to
disk & update
disk & draw
addresses

copied
all system
library?

initialize

LDRES

initialize
all quantities in
the swap
area for this
user number

fix this guy's
TTY table too

write out
his swap area

have we done
all of them?

LDRES

LDRES
This routine checks for the first reel of a mag tape set.

TPLR

TPLRA

Rewind the mag tape

Read first record on tape

Is record a tape label?

Yes

Is this reel number one?

Yes

Return

No

Print 'Tape reel #1, delete end'

Halt

TPLRA
ROUTINE TO GET DISK SPACE DURING INITIAL OR MAINT LOAD.

CDB

Read in disk # ADT and initialize variables

SET UP FOR 5 SYSTEM SEGMENTS

Fetch space from this segment NO ROOM ON DISK

Print "INSUFFICIENT DISK SPACE"

ERRIN

Advance pointers, deal with all fields

GET SPACE FOR FIRST DAT ENTRY (SYSTEM LIBRARY)

NO ROOM ON THIS DISK

RESTORE CURRENT ADT, FETCH NEXT

GET SPACE FOR ADT SBT TRAX

GET SPACE FOR DISK ADS.

GET SPACE FOR DIRECTORY TRAX

RESTORE ADT TO BLM.

COPY BLM TO DISK

RETURN

THE CORR ROUTINE IS USE FOR THESE.
SVSL8

SYSLB

First call on paper tape load

RETURN

is switch IS up?

HLT ISy

Is this the first system library prog.?

set up initial core, disk, & drum pointers.

Get word set for disk & drum or next record on this drum?

advance to next drum track.

write to disk or drum

advance both disk and drum addresses

RETURN
The specified first block lies between four entries.

CPB: will this entry combine with the proceeding one?

Yes → MLK12

No → Insert a new entry.

MLK12

Check for overlap of following entries.

Entry is after all other entries in table.

MLK18

CPB: will this entry combine with the proceeding one?

Yes → MLK19

No → Insert a new entry.

MLK49
LDLAC

ASK CONFIGURATION OPTIONS?

YES

BUADT

BUILD A BRAND NEW ADT

COPY THE DISK AIDS TO THE DISK

CLUDT

GET THE REMAINDER OF DISK TRACKS

COPY THE TRD TRACKS TO THE DISK

COPY THE DISK TRACKS TO THE DISK

SRC
SLDM

Initialize variables and initial locations

WDLTD
write out updated final disk bootstraps

SDRT
copy drum tables to the disk

write out the
DRT & DIREC

write out the
new EQUIPMENT
TABLE

beginning of
sleep routine in
loader.
Magnetic tape sleep & hibernate.

Is a MAC tape present ?
yes
no

Initialize for special entry

TENCH special entry to TENCH to write first tape label

DUMP THE EOT & DIREC tables

CMIDR dump the 3D table

CMIDR dump the DIRECTORY

Dump the system segments

dump all of the library routines

EXIT
DSKC

Move in new end pseudo-entry

DSRF

Add 12 to the correct word count

DSKC

Fix error message for DSR and end

Print error message

DSPA

Find the new length for the DAT and update the DSRF table

DSRF
TENCH

- If this a write or a verify must
- verify
- are the save variables identical to start?
  yes
  return
  no
  write

- verify file mark, trailer label, and following file mark

Routine to handle end of tape while writing.

Note: Returns from this routine are always to the address presently in the save zero area. Since this save area may be changed by this routine, returns are not always to the immediately preceding call.
BEGIN

REWARD AND STAND BY

IS THIS AN END OF SET?

YES
RETURN

NO

ADVANCE REEL NUMBER AND PRINT "MOUNT REEL NUMBER XX, PRESS RUN."

HLT 77B

SAVE CURRENT AREA ONE TO SAVE AREA ONE AND TWO

SET DRIVES TO WRITE AND REWIND THE TAPE

WRITE THE NEW PAGE LABEL

RETURN
LOAD a DUMP

START

IS there anything to do? no = LOAD

MAG Tape Present? no = GMTS

configure the drive & rewind the tape.

LOAD or Dump? no = Dump

copy the modified directory to the disk

take entry to sleep at MDS E

TPLR demand a reel number

SPFS
This routine implements end of tape label and presents a continuous motion to the calling routine.

- **MTRD**
  - **Is a record already in?**
    - No: **MTRD**
    - Yes: **MTRD**
  - **Is it an end of file?**
    - No: **MTRD**
    - Yes: **MTRD**
  - **Read next record from tape.**
    - **Is record a label?**
      - No: **MTRD**
      - Yes: **MTRD**
    - **Same record into set flag to say record present.**
    - **Give EOF**
      - **RETURN**
Bootstrap procedure

start BBDL at 777608

BBDL       reads in drum sector 0

Sector 0   reads in drum sector 1

Sector 1   reads in disc blocks 1 & 2
NOTE: this loader is configured to read from unit 0
of the disc controller in select code 17B.

Disc block 1 & 2 this block scans all disc drives and builds a
table of drives vs. logical unit number. This bootstrap
contains the system segment table and uses it to read in
the system.
Sleep Tape Format

The following lists the records on a sleep tape in order starting from the load point marker.

Tape Label

The tape label is a seven word record and appears at the beginning and end of each sleep tape. The words are:

0    ASC 1, LB
1    ASC 1, TSB
2    (unused)
3    (reel number)
4    (year e.g. 71)
5    (hour of year)
6    (tenth of seconds)

EQT

\~ 120B words

DIREC

\~ 560 words

1DT

1024 word blocks, the 1DT and Directory are packed into 1024 word blocks as they are read in, without regard to track or entry boundaries.

Directory

1024 word blocks

see 1DT format
System Segment Table

System

The system is written out according to the information in the System Segment Table.

System Library

Each system library program is dumped in a separate record.

EOF

This end of file ends the system information. It must be on the first reel of tape.

USER Library

Programs and Files are written out in 1024 word records (4 blocks. Files remain interleaved. i.e. the first tape record of a 2n
block long File contains records 1, n + 1, 2, n + 2). The First tape record contains the 12 word directory entry at its beginning and may be up to 1036 words long. A File mark is written after the last tape record of each program or File.

EOF

An EOF in place of an initial record of a program or File indicates the end of the sleep set.

NOTES:

1. When an end of tape marker is detected, the mag tape dump will write an EOF, a label record, and another EOF and ask for another reel. During reading, the sequence of an EOF followed by a label (7 word) record will indicate the end of a reel. No other test for end of reel is possible.